НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис) (ім'я, прізвище)

«___»_____ 2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою *"Комп'ютерні системи та мережі"*

спеціальності *123 "Комп'ютерна інженерія"*

на тему *"Спосіб маршрутизації в інтелектуальних мережах з технологією SDN"*

Виконала: студентка *IV* курсу, групи *ІО-61*
Осієвська Валентина Сергіївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *асистент Калюжний Олександр Олегович* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант н. контроль *доц, д. т. н. Сімоненко В. П.* _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент *доц. каф. СПіСКС к.т.н., доц. Орлова М. М.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ - 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

Рівень вищої – освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма – «Комп'ютерні системи та мережі»

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО
(підпис) (ім'я, прізвище)

«___»_____ 2020 р.

ЗАВДАННЯ

на дипломний проект студентці

Осієвській Валентині Сергіївні

1. Тема проекту *"Спосіб маршрутизації в інтелектуальних мережах з технологією SDN"*, керівник проекту *Калюжний Олександр Олегович асистент* , затверджені наказом по університету від *« 07 » травня* 2020р. № 1081-с

2. Термін подання студентом проекту *25.05.2020р.*

3. Вихідні дані до проекту: *технічне завдання, теоретичні дані, наукові публікації.*

4. Зміст пояснювальної записки: *огляд методів маршрутизації інтелектуальних мереж, конструювання трафіку в програмно визначених транспортних мережах, проектування та розробка програмного забезпечення, тестування програмного модулю та аналіз отриманих результатів.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання 01.09.2019 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1	*Затвердження теми роботи*	*01.09.2019*	
2	*Вивчення та аналіз завдання*	*14.09.2019*	
3	*Розробка архітектури та загальної структури систем*	*12.10.2019*	
4	*Розробка структур окремих підсистем*	*26.10.2019*	
5	*Програмна реалізація системи*	*30.11.2020*	
6	*Оформлення пояснювальної записки*	*28.03.2020*	
7	*Передзахист*	*26.05.2020*	
8	*Захист*	*25.06.2020*	

Студентка Валентина ОСІЄВСЬКА _____
(підпис)

Керівник Олександр КАЛЮЖНИЙ _____
(підпис)

Анотація

Бакалаврська дипломна робота присвячена вирішенню проблеми конструювання трафіку в інтелектуальних мережах за допомогою технології SDN. Запропонований спосіб маршрутизації зорієнтований на використання переважно у транспортних мережах. Описаний програмний продукт створює можливість скорочення матеріальних витрат та аварійних ситуацій, підвищує надійність системи та швидкість конструювання трафіку.

Annotation

The Bachelor's thesis is developed for solving the problem of traffic engineering in intelligent networks, using SDN technology. The proposed routing method is focused on use mainly in transport networks. The described software product creates the possibility of reducing material costs and emergencies, increases system reliability and speed of traffic engineering.

№ рядка	Формат	Позначення	Найменування	Кільк. аркушів	№ прим.	Примітка
1	А4		Завдання на дипломний проект	2		
2	А4	ІАЛЦ.466454.001 ВД	Відомість дипломного проекту	1		
3	А4	ІАЛЦ.466454.002 ТЗ	Технічне завдання	3		
4	А4	ІАЛЦ.466454.003 ПЗ	Пояснювальна записка	63		
5	А4	ІАЛЦ.466454.004 Д1	Структурна схема пристрою	1		
6	А4	ІАЛЦ.466454.004 Д2	Функціональна схема	1		
7	А4	ІАЛЦ.466454.004 Д3	Принципова схема апаратного забезпечення	1		
8	А4	ІАЛЦ.466454.004 Д4	Програмний код додатку	10		

Зм.	Арк.	№ документа	Підпис	Дата	ІАЛЦ.466454.001.ВД		
Розробив		Осієвська В. С.			Способи маршрутизації в інтелектуальних мережах з використанням технології SDN Відомість дипломного проекту	Лит.	Лист. 1 / Аркушів 1
Перевірив		Калюжний О. О.					
Реценз.						НТУУ "КПІ", ФІОТ, ІО-61	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Спосіб маршрутизації в інтелектуальних мережах з технологією SDN»

Київ – 2020 року

ЗМІСТ

1. Найменування та область застосування

Розробка програми для реалізації способу маршрутизації в інтелектуальних мережах з використанням технології SDN. Область застосування — використання мережі Інтернет.

2. Підстава для розробки

Підставою для розробки є індивідуальне завдання на реалізацію алгоритму маршрутизації в інтелектуальних мережах з використанням технології SDN, затверджене кафедрою Обчислювальної техніки ФІОТ НТУУ “КПІ ім. Ігоря Сікорського”.

3. Призначення розробки

Даний проект призначений для реалізації алгоритму маршрутизації в інтелектуальних мережах з використанням технології SDN, а також для демонстрації набутих навичок та закріплення отриманих знань.

4. Джерела розробки

Основними джерелами розробки є науково-технічна література по програмному забезпеченню, наукові публікації про конструювання трафіку в інтелектуальних транспортних мережах та результати застосування програмно-забезпечуваних мереж при конструюванні трафіку.

5. Технічні вимоги

5.1 Вимоги до розроблюваного продукту

- Розробка інтерфейсу користувача для ручного введення даних та виведення результатів аналізу
- Візуалізація мережі у вигляді зваженого графу з позначенням шляхів визначених в результаті аналіз

- Виконання симуляції роботи адаптованої системи програмним шляхом для реалізації алгоритму пошук

5.2 Вимоги програмного забезпечення

- Ubuntu Linux 14.x+, Mac OS 10.8.3+, 10.9+, MS Windows 10
- Java SE 8+
- Інтеграційне середовище розробки IntelliJ IDEA, Eclipse

5.2 Вимоги апаратного забезпечення

- Для Windows: RAM 128 МБ, 128 для JRE, 2 МБ для оновлення java; мінімальна вимога до процесора Pentium 2 266 МГц
- Для Mac OS X: Mac на базі процесора Intel, браузер Chrome, Safari
- Для Linux: браузер Chrome, Safari

5.3 Вимоги до надійності

- Написання тестів для програмного забезпечення
- Проходження тестування у процесі роботи системи
- Строк використання — необмежений

6. Етапи розробки

1.	Дослідженні існуючих рішень	01.09.2019
2.	Постановка та аналіз технічного завдання	14.09.2019
3.	Розробка архітектури та основних модулів системи	12.10.2019
4.	Тестування окремих модулів та усієї системи	26.10.2019
5.	Налагодження програмного продукту	30.11.2020
6.	Написання документації до дипломної роботи	28.03.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Спосіб маршрутизації в інтелектуальних мережах з технологією SDN»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IM	Інтелектуальні мережі
IVN	(Intelligent Vehicle Network) Інтелектуальна мережа транспортних засобів
VANET	(Vehicular ad-hoc networks) Спеціальні транспортні мережі
GPS	(Global Positioning System) Система глобального позиціонування
RSU	(Road Side Unit) Розподілена система управління
SDN	(Software-defined networking) Програмно-конфігурована мережа
SDVN	(Software-defined vehicle networking) Програмно-конфігурована транспортна мережа
API	(Application Programming Interface) Прикладний програмний інтерфейс

ВСТУП

На сьогоднішній день спостерігається швидкий розвиток комп'ютерних мереж. Основною причиною пошуку нових, більш вдосконалених рішень, є стрімке зростання кількості користувачів. Розширення спектру технічних можливостей та послуг, створюваних користувачем, встановлюють нові вимоги до мереж та ускладнюють їх подальше застосування та управління. Нові, більш високі вимоги до потужності та ємності вирішуються шляхом перерозподілу мережі та розбиття вузлів, що на практиці призводить до трудомісткого та дорогого перенастроювання мережевих пристроїв на місці. Технічне обслуговування мережі традиційно є дуже трудомістким завданням, і реалізація такого рішення може підвищити ризик виникнення людського фактору, що призведе до зниження показника надійності мережі та збільшення показника експлуатаційних витрат. Таким чином, поряд із задачею надання необхідних послуг, постає задача пошуку рішень оптимізації та вдосконалення реалізованої системи.

Елегантним способом зменшити дію людського фактору на робочу систему є наділення мережі інтелектом. У даному випадку, під "наділення інтелектом" варто розуміти інтелектуальні мережеві пристрої, що використовуються для керування програмним забезпеченням, контролю та моніторингу всієї мережі. Особливо відзначаються пристрої здатні до автоматичного самоналаштування, визначення проблеми та прийняття рішень її усунення. Використання інтелектуальних мереж демонструє вражаючі результати у вигляді економії витрат на експлуатацію та надійності систем. Тим не менш, для вирішення певних проблем все ще залишається необхідним використання додаткових технологій для вирішення певних проблем.

Метою даної роботи є реалізація методу маршрутизації в інтелектуальних транспортних мережах. На сьогодні, використання

спеціальних транспортних мереж є одним із перспективних напрямів. VANET встановлює мобільну мережу між транспортними засобами, що рухаються, розглядаючи кожен транспортний засіб як вузол у мережі. Спілкуючись один з одним ці вузли утворюють Інтелектуальну мережу транспортних засобів (IVN), яка є одним із важливих елементів розумних міст.

Порівняно зі статичними або низькошвидкісними рухомими вузлами в традиційній бездротовій мережі, транспортні засоби рухаються швидше та непердюачувано, внаслідок чого часто змінюються налаштування мережі VANET. Оскільки, більшість відомих методів маршрутизації залишаються не ефективними, пошук оптимальної методики руху трафіку з урахуванням особливостей транспортних мереж залишається актуальною задачею. Одним з перспективних підходів до вирішення цієї проблеми є інтеграція спеціальних транспортних мереж з визначеними програмним забезпеченням мережами.

Об’єктом дослідження даної роботи виступають інтелектуальні транспортні мережі.

Предметом дослідження є реалізація способу маршрутизації з використанням інтелектуальних мереж.

РОЗДІЛ 1. ОГЛЯД МЕТОДІВ МАРШРУТИЗАЦІЇ ІНТЕЛЕКТУАЛЬНИХ МЕРЕЖ

1.1 Інтелектуальної мережі

Термін "Інтелектуальна мережа" (ІМ) використовується для опису архітектурної концепції, що застосовується до всіх телекомунікаційних мереж. Основною метою IN є полегшення впровадження та надання нових послуг базуючись на більшій гнучкості та спектру нових можливостей, серед яких: універсальні приватні телекомунікації, віртуальна приватна мережа, безкоштовний телефон, тощо.

Надання переваги використанню ІМ зумовлене прагненням постачальників телекомунікаційних послуг до покращення якості та зменшенню витрат на операції по управлінні мережею. Економічні витрати мають ефективно та диференційно задовольняти існуючий та потенційний попит послуг на ринку. Розвиток сучасних тенденцій в галузі технологій дозволяє підвищити рівень інтелекту в телекомунікаційній мережі, і тим самим задовільнити потреби постачальників. Наприклад, покращена мобільність, отримана внаслідок зменшення розмірів електронних компонентів дозволяє досягти більшого ступеня розподіленої функціональності всередині самих компонентів та, безпосередньо, між ними.

Завдання ІМ полягає в тому, щоб дозволити включення додаткових можливостей для полегшення надання послуги, незалежно від реалізації послуги в середовищі для багатьох постачальників.

1.1.1 Принцип роботи інтелектуальної мережі

Основа роботи інтелектуальних мереж базується якісному на впорядкуванні та створенні нових сервісів, а також контролю старих сервісів задля збільшення попиту саме на ці систему серед клієнтів. Перш за все послуги організовані у вигляді спеціальних типів послідовностей для належного функціонування. Воно здійснюється за допомогою спеціальних будівельних блоків, які називаються службовими незалежними будівельними

блоками (SIB). Ці блоки допомагають ініціювати різні типи сповіщень та оголошень. Після цієї запуску інтелектуальної мережі необхідно контролювати загальну роботу системи та впроваджувати нові послуги мережевої системи, встановлюючи блоки двох службових ліній, таких як дві телефонні лінії. Функція цього етапу - давати вказівки для розмови, яка ведеться під час дзвінка. Основним компонентом інтелектуальної мережі з погляду оператора є SMS (System Management System). Більшою мірою, головною метою системи є управління точками доступу, а також участь у виставлення рахунків. Ще одним важливим компонентом інтелектуальної системи є сервісні точки управління (SCP); він управляє різними видами послуг між лінією та замовником, а також перевіряє автоматизовані оголошення. Різні типи інших компонентів інтелектуальної мережі також беруть участь у робочій, тобто SCF (функція управління сервісом) або SDF (функція бази даних служби)

1.1.2 Архітектура інтелектуальної мережі

Інтелектуальна мережа (IM) виникла в середині 1980-х років як спосіб від'єднання логіки послуги телекомунікацій від функцій комутації викликів бірж. Така інновація полегшила функціональність централізованої обробки послуг, розгортання нових сервісів та зменшила складність обмінів. Стандартизація IM була реалізована в ANSI, ITU та ETSI. На жаль, такий розподіл зусиль щодо стандартизації та фірмових покращень створили для постачальників послуг безліч рішень без належної взаємодії між собою. У таких регіонах як США чи Європа, існує певна домовленість відносно стандартів можливої взаємодії. Стандарт MCE є основою для міжнародної взаємодії.

Основна модель IN включає в себе розподілену функціональну архітектуру, яка містить функціональні об'єкти, які несуть колективну відповідальність за обробку будь-яких дзвінків, та використовують більш

складні послуги, ніж традиційна маршрутизація викликів набраними цифрами. На рисунку 1 показано функціональні об'єкти, їх зв’язки та те, як ці об'єкти зазвичай групуються в платформи в мережі. Усі відносини управління зазвичай підтримуються сигналом, що передається через надійну мережу пакетних даних SS.7 (Система сигналізації № 7)

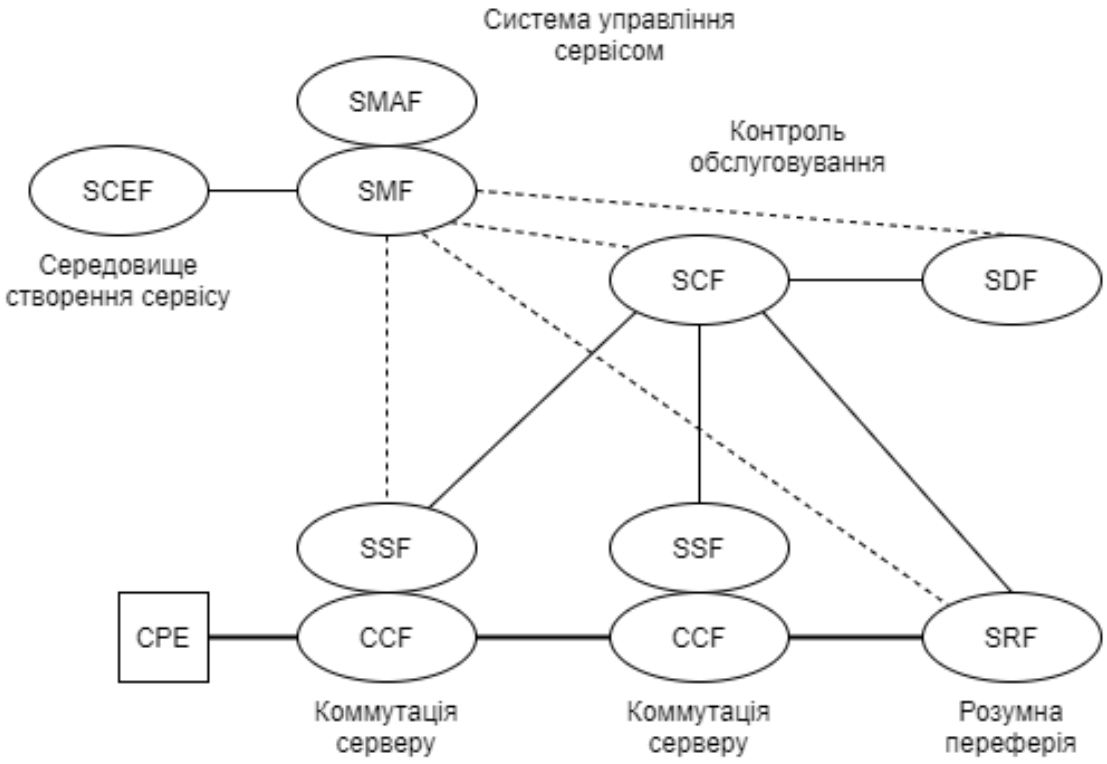


Рис. 1.1. Архітектура інтелектуальної мережі

SMF – функція управління сервісом;

SMAF – функція агента управління сервісом

SCEF – функція середовища створення служби

SRF – функція спеціалізованих ресурсів

SCF – функція управління сервісом

SSF – функція комутації послуг

SDF – функція службових даних

CCF – функція управління викликом

CPE – обладнання приміщень замовника

Кожна функція відіграє певну роль у завершенні викликів, які потребують додаткової обробки для маршрутизації через мережу. Головною задачею CCF є сонтороль маршрутизації звичайних дзвінків через PSTN і визначення, коли слід розпочати сеанс інтелектуальної маршрутизації, наприклад, вивчаючи набрані цифри. Для виклику IM CCF надає контроль колокованому SSF, який запускає сеанс INAP (Intelligent Network Application Protocol) з SCF, щоб мати змогу створити запит на будь-які спеціальні інструкції з обробки цього дзвінка. Зазвичай SCF буде відповідати адресою (номером) призначення, на який слід переадресувати виклик. Дана схема є базовою та відображає можливість виникнення великої кількості варіантів, включаючи запит SCF до бази даних (SDF) або вказівку SSF підключити виклик до пристрою відтворення запису в SRF. Функція SCF може також доручити SSF щодо конкретних схем виставлення рахунків, які повинні використовуватися при вирішенні цього дзвінка. суб'єкти, що надають послуги, надають такі функції, як створення сервісу (SCEF) та управління послугами (SMF, SMAF).

1.1.3 Переваги та недоліки інтелектуальних систем

Важливі переваги інтелектуальних мереж:

- однією з головних переваг інтелектуальних мереж є оцифровка та зміна лінії комутації;
- інтелектуальні мережі керують усією системою або сервісами телекомунікацій, а також контролюють всю систему, з подальшим запуском необхідних програм;
- інтелектуальні мережі сприяють росту конкуренції на ринку, розробляючи та впроваджуючи нові послуги.

Серед недоліків IM можна виділити наступні:

- інколи для підтримання надійності постачальники послуг вимушені використовувати неінтелектуальні мережі;

- складна конфігурація також є недоліком інтелектуальних мереж, оскільки вона створюється за допомогою невеликих апаратних засобів;
- операторам потрібні великі витрати на встановлення або конфігурацію інтелектуальних мереж.

1.2 Інтелектуальні транспортні Мережі

Останнім часом, завдяки постійно зростаючій кількості транспортних засобів на дорогах, автомобільні спеціальні мережі стають одним із найперспективніших наукових напрямів [1]. VANET встановлює мобільну мережу між транспортними засобами, що рухаються, розглядаючи кожен транспортний засіб як вузол у мережі. Спілкуючись один з одним ці вузли утворюють Інтелектуальну мережу транспортних засобів (IVN), яка є одним із важливих елементів розумних міст.

Зі швидким збільшенням кількості транспортних засобів та обмеженою площею дорожньої інфраструктури транспортна індустрія стикається з різними проблемами, такими як збільшення аварій на дорожньому транспорті, тривалі перевезення автомобілів, пошкодження суспільної власності та людських життів тощо. Для подолання цих проблем та для підвищення ефективності транспортної системи, з'явилася нова парадигма під назвою спеціальна транспортна мережа (VANET).

На відміну від статичних систем VANET володіє деякими унікальними характеристиками, такими як передбачувані моделі мобільності, великий розмір роботи в мережі, часті відключення, швидкість зміни топології та жорсткі обмеження затримки. Незважаючи на те, що дослідження VANET тривають майже десятиліття, вони все ще стикаються з багатьма проблемами, такими як поширення неефективної інформації, проблеми безпеки та конфіденційності, незбалансований трафік, нестабільне використання ресурсів, відсутність якості тощо [2].

1.2.1 Архітектура спеціальних транспортних мереж

VANET складається із зв’язків “транспортний засіб” – “транспортний засіб” (V2V) та між інфраструктурних зв’язків (I2I) [3]. Як правило, ці пристрої будуть оснащені пристроями GPS та спеціалізованими радіостанціями короткого діапазону зв'язку. 2У спеціальних транспортних мережах придорожні пристрої (RSU) спеціалізуються на обміні інформації між транспортними засобами, а базові станції, підключені до RSU, забезпечують доступ до мережі, як показано на рис. 1.2.

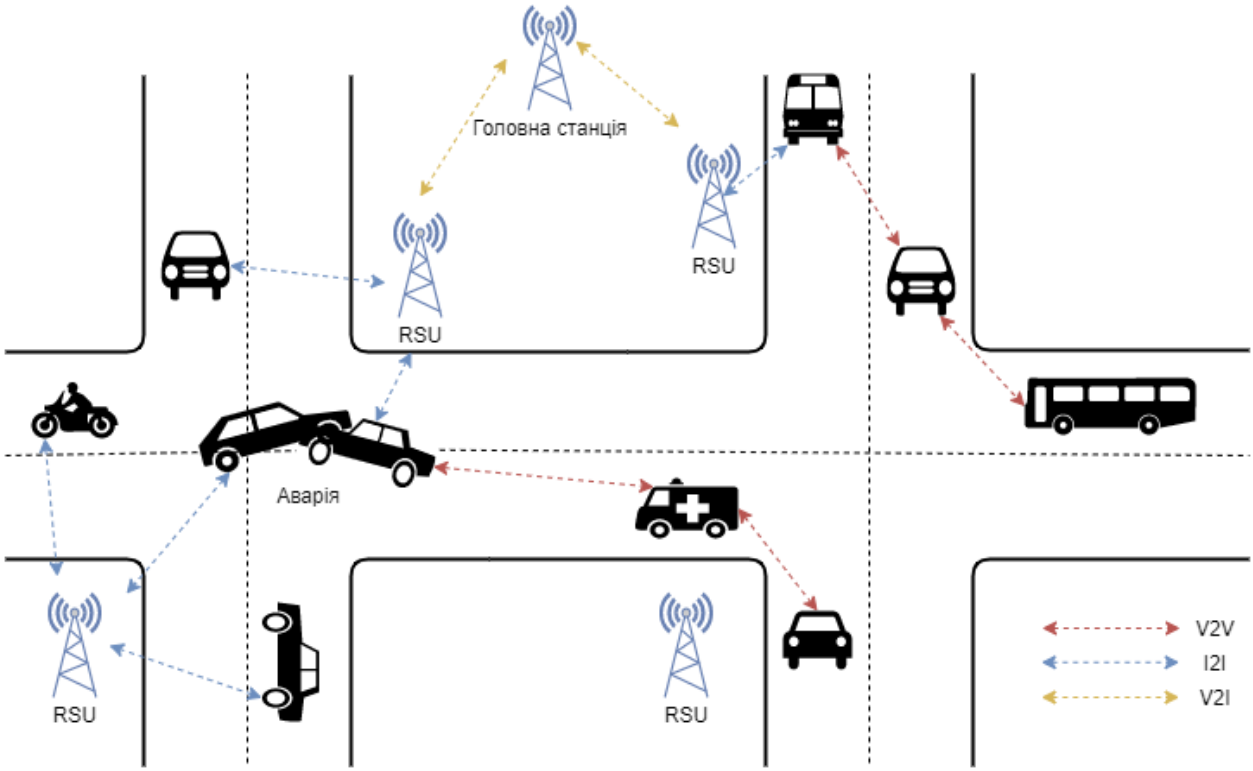


Рис. 1.2. Базова архітектура транспортних мереж

У наш час ця система суттєво розширила свою сферу застосування, і тепер вона охоплює поширення та обмін інформації між різноманітними транспортними засобами, створення інтернету транспортних засобів, забезпечення дорожнього керівництва, допомоги під час паркування, тощо.

1.2.3 Протоколи маршрутизації спеціальних транспортних мереж

Враховуючи такі особливості транспортних мереж, гостро постає питання вибору правильного протоколу маршрутизації. Складність передачі

пакетів від одного транспортного засобу до іншого, полягає у тому, що вони рухаються з різною швидкістю, періодично змінюють свій напрям, то зменшуючи то збільшуючи відстань між ними. На рис. 1.3 наведена класифікація відомих протоколів маршрутизації мережі VANET.

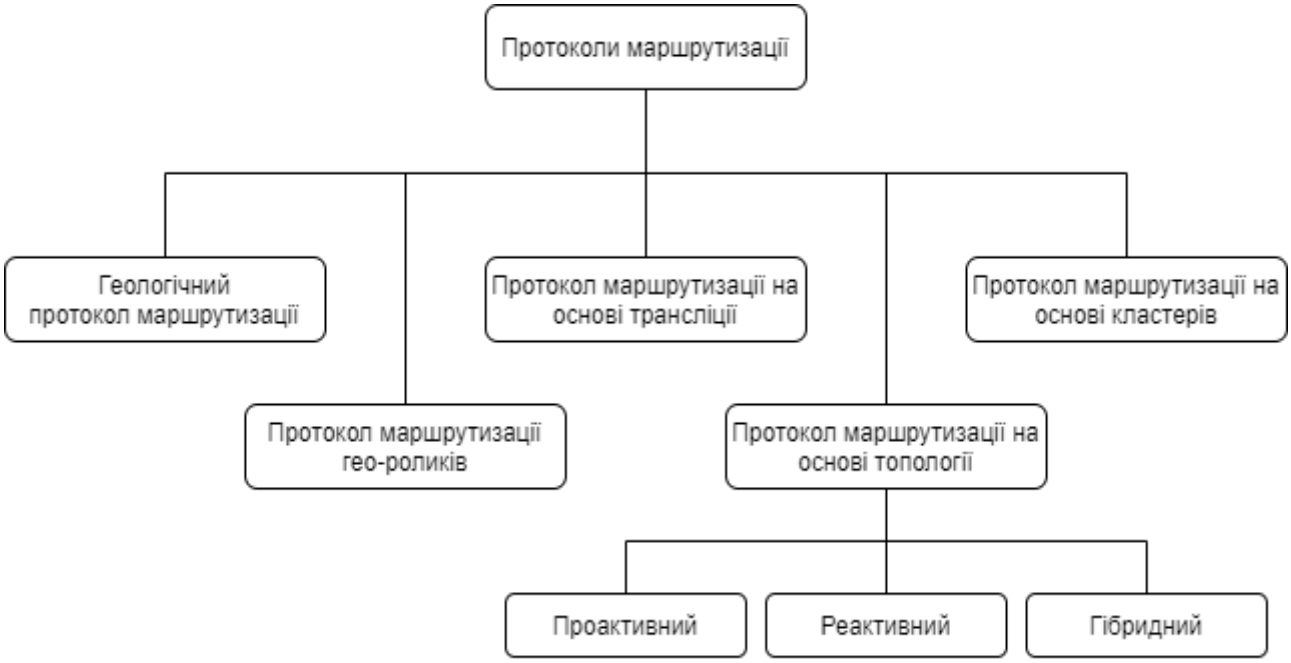


Рис. 1.3. Класифікація протоколів маршрутизації мережі VANET

- *Геологічний протокол маршрутизації*

Визначною рисою таких протоколів є комунікування вихідної та кінцевої точки за допомогою географічних положень та мережевої адреси. Прикладом служить протокол балансування завантаження (LBRP), який проводить обчислення та налаштування маршруту беручи до уваги інформацію отриману на основі розташування вузлів. Такий підхід не потребує побудову таблиць маршрутизації. Як правило, LBRP складається з трьох компонентів, а саме: маяк, локація та послуги переадресації. Превагою таких протоколів є ефективність роботи у високому мобільному середовищі. Проте цей протокол, як і інші має свої недоліки, наприклад, для визначення місця розташування транспортних засобів LBRP потребує допомоги системи глобальної позиціонування (GPS). Якщо розглядати стабільну систему,

наприклад швидкісне шосе, то використання такого протоколу дає доволі високі показники, проте, якщо маршрут транспортного засобу не є стабільним, і автомобіль буде пересуватись по тунелю чи під землею, супутникові сигнали будуть відповідно слабшати. Прикладом таких протоколів служить жадібний алгоритм ефекту маршрутизації без обмеження стану та відстані для маршрутизації (DREAM) [4].

- *Протокол маршрутизації на основі трансляції*

Даний протокол використовується здебільше, коли стартовий вузол та кінцевий вузол знаходяться за межами один одного. В основному вони використовуються у додатках, призначених для збереження безпеки, таких як: попередження про погодні умови чи стан транспортного руху, попередження аварійних ситуацій, тощо. Такі протоколи записують у пакет дані для усіх мереж VANET усім доступним вузлам у широковому домені. Прикладом є протокол надійного мовлення (DECA), зорієнтований на щільність, визначною рисою якого є надійність. Недоліком є те, що ці типи протоколів споживають більшу пропускну здатність, і на кінечний вузол приходить велика кількість повторюваних пакетів, що не добре впливає на роботу протоколу [5].

- *Протокол маршрутизації на основі кластерів*

Особливістю цього протоколу є групування транспортних засобів за характеристиками швидкості, напрямку руху, тощо в окремий кластер. Таким чином, якщо два транспортні засоби знаходитимуться в одному кластері та матимуть потребу зв’язатись між собою в середині кластера, такий зв’язок буде називатись локальним зв’язком. Крім того, якщо виникає потреба зв’язати два вузли виходячи поза межі одного кластера, якщо вузол транспортного засобу повинен зв’язати вузол, то для досягнення пункту призначення необхідна допомога головного кластера. Зважаючи на фактор масштабованості, протокол маршрутизації на основі кластерів є хорошим

вибором для дизайнерів мережі. Негативною рисою є затримка руху. Прикладом такого протоколу є кластеризація для відкритої міжмобільної мережі зв'язку (COIN) [6].

- *Протокол маршрутизації* Geo-cast

Протокол Geo-cast складається з зони актуальності (ZOR) та зони переадресації (ZOF). Зона актуальності представляє собою тереиторію, визначену для вузлів одного регіону. Як раз таки створення можливості слілкування транспортних засобів, що надежать одній ZOR і є головною метою протоколу Geo-cast. Згідно цього протоколу, якщо стартовий транспотний засіб матиме за необхідність зв’язатись з іншим транспортним засобом, який не знаходиться в ZOR першого транспортного засобу, він стане частиною ZOF, і будь-який транспортний засіб, який входить до ZOF, стане відповідальним за пересилання інформації до інших ZOR. До недоліків такої системи відноситься часте переривання з’єднання [7].

- Протокол маршрутизації на основі топології

Серед протоколів маршрутизації, основаних на топології, можна виділити три основні типи протоколів, такі як: проактивний, реактивний та гібридні протоколи [8].

a. Проактивні протоколи

Проактивні протоколи виконують регулярні оновлення таблиці маршрутизації у зв’язку з постійними обчисленнями маршрутів. Серед таких протоколів великої популярності набуває алгоритм Bellman Ford, оскільки він дозволяє вузлам зберігати інформацію про наступні вузли. У цьому полягає основна перевага проактивного протоколу: за необхідності передачі пакетів даних, маршрут буде уже відомий. Прикладами цього протоколу виступають оптимізований протокол маршрутизації стану зв’язку (OLSR) та протокол призначення послідовного дистанційного вектора (DSDV) [9].

b. Реактивні протоколи

Цей тип протоколів є протилежністю проактивного протоколу та не зберігає інформації про усі вузли мережі. Актуальною вважається лише та інформація, про вузли, які надходять у маршрут. Популярними прикладами таких протоколів маршрутизації служать спеціальний векторний протокол по запиту (AODV), протокол динамічного керування джерелом управління (DSR) та динамічний протокол Manet on Demam (DYMO) [10].

c. Гібридні протоколи

Відповідно до своєї назви, гібридні протоколи — це поєднання проактивних та реактивних протоколів маршрутизації. За рахунок такої модифікації ми отримуємо можливість зменшити накладні витрати та затримки. Реалізації такого підходу, необхідно реалізувати обмін інформацією про топологію. Використання гібридного підходу підвищило показники ефективності та масштабованості мережі. Безперечно, гібридного підходу має і свої недоліки, такі як, висока затримка навігації по нових маршрутах. Загальний протокол, заснований на гібридному підході, - протокол маршрутизації зон (ZRP) [11].

1.3 Програмно-визначені мережі

Метою програмно-визначених мереж є забезпечення гнучкості та підвищення швидкості роботи мережі, вдосконалення мережевого контролю, створюючи можливість для підприємств та постачальників послуг оперативно реагувати на швидкозмінні правила та вимоги бізнесу.

Особливість мереж визначених програмним забезпеченням полягає у більш вдосконаленому методі конструювання трафіку. SDN створює для адміністраторів можливість формувати трафік з централізованої консолі управління, без ручного втручання в роботу мережевих пристроїв. У технології SDN, централізований контролер керує комутаторами для надання мережевих служб там, де вони потрібні, незалежно від встановлених з'єднань

сервера та елементів мережі. Така специфіка роботи системи є певним протиставленням традиційної мережевої архітектури, в якій за роботу трафіку відповідають конкретні мережеві елементи базуючись на власних налаштуваннях відповідно до таблиці маршрутизації.

1.3.1 Архітектура SDN

У базовій архітектурі технології SDN можна виділити три основні рівні, такі як: прикладний рівень, керуючий рівень та інфраструктурний рівень. ЇЇ графічне відображення ви можете побачити на рисунку 1.4 [12].

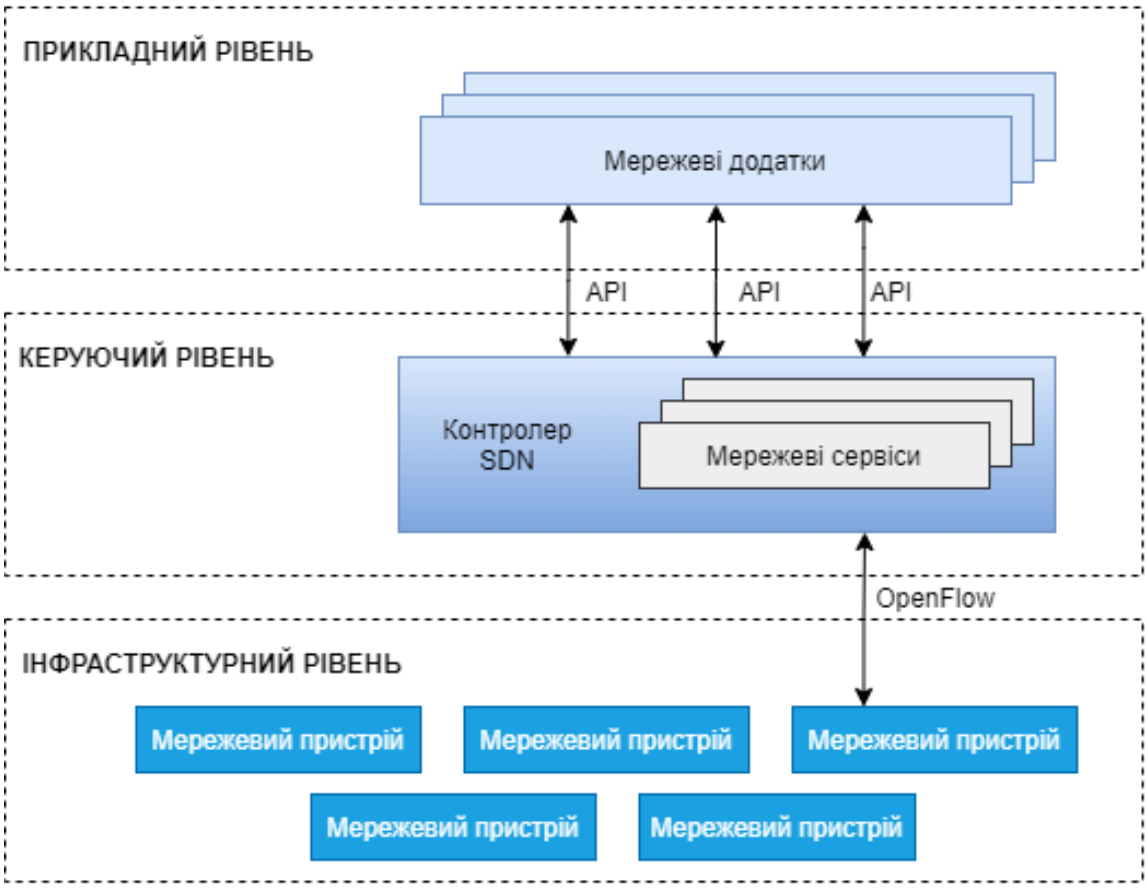


Рис. 1.4. Архітектура програмно-визначених мереж

На прикладному рівні зберігаються стандартні мережеві додатки або функції. Такі мережеві додатки використовуються організаціями для запуску систем виявлення помилок або вторгнень, балансування, навантаження трафіку або брандмауери. На відміну від традиційних мережі, програмно конфігуровані мережі можуть замінити мережевий пристрій додатком, який

контролер буде застосовувати для управління поведінкою площини даних. В архітектурі SDN мережу можна умовно розділити на три розподілені рівні, що з'єднуються через північний та південний API.

Основою керуючого рівня є централізоване програмне забезпечення контролера SDN. Таке ПО виконує мозкову функцію в життєвому циклі мережі. Контролер SDN розташований на сервері і виконує управління конструювання та управління потоком трафіку усієї мережі.

Множина мережевих пристроїв утворює інфраструктурний рівень SDN.

Комунікація між трьома шарами реалізовується за допомогою відповідних інтерфейсів прикладного програмування (API). Обмін інформації можливий в обох напрямах, північний та південний. Таким чином, програми або додатки проводять спілкування з контролером через його північний інтерфейс, у той час, як контролер та комутатори спілкуються за допомогою відповідних інтерфейсів на південному API, такому як OpenFlow, хоча існують й інші протоколи.

На сьогоднішній день не можна виділити офіційного стандарту для API-контролера на північному рівні, який би відповідав OpenFlow як загальному південному інтерфейсу контролера. Імовірність того, що північний інтерфейс API контролера OpenDaylight може стати фактичним стандартом, беручи до уваги його підтримку від постачальників.

1.3.2 Принцип роботи SDN

Використання SDN створює можливість застосування різних типів технологій, таких як функціональне розподілення, віртуалізація мереж та програмування автоматизації.

На початку технологія SDN була головним чином спрямована на розділення керуючого та прикладного рівня. Тобто, у той час, як керуючий рівень зайнятий пересиланням пакетів по мережі, прикладний рівень фактично

реалізовує переміщення пакетів з місця на місце. Згідно стандартного сценарію, пакет SDN потрапляє на мережевий комутатор, який, в свою чергу, завдяки стеку правил вбудованому у власну прошивку, приймає рішення, куди цей пакет має пересилатись у майбутньому. Такий стек правил отримується комутатором напряму від централізованого контролера.

Комутатор також можна також ідентифікувати, як пристрій площини даних. Його функцією є запит на управління в контролера та отримання безпосередньої інформації про трафік. Комутатор відповідає на такий запит та надсилає всі пакети за місцем призначення по визначеному шляху. Існує режим роботи, у якому комутатор відповідає на запит контролера пакетом, який не має певного маршруту. Такий режим роботи називається адаптивним, або динамічним. У такому випадку комутатор видає запити маршрутів через маршрутизатори або алгоритми на основі мережевої топології.

1.4 Програмно-визначені транспортні мережі

Використання технології SDN у транспортних мережах обумовлене необхідністю спрощення роботи існуючих мереж, оптимізація ефективності маршрутизації викликів та підвищення продуктивності роботи мережі. SDVN поєднує у собі глобальне рішення конструювання трафіку та керування цілою мережею. Своєї популярності вони набувають за рахунок надання можливості розгортання та тестування нових служб та протоколів, а також можливості впровадження нових додатків. У результаті чого, ми маємо нову систему SDVN, яка забезпечує швидке впровадження інновацій у сфері комп'ютерних мереж. Завдяки своїй унікальній архітектурі, різного типу бездротові пристрої, у тому числі спеціальні мобільні мережі, транспортні мережі та будь-які придорожні бездротові засоби зв'язку, вважатимуться окремим комутатором SDN та матимуть власний ідентифікатор та інтерфейс. Крім того, більшість мережевих ресурсів такі як спектр або межа пропускання можуть так само

керуватись безпосередньо централізованою площиною управління. Досліджувана архітектура технології SDVN зображена на рисунку 1.5.

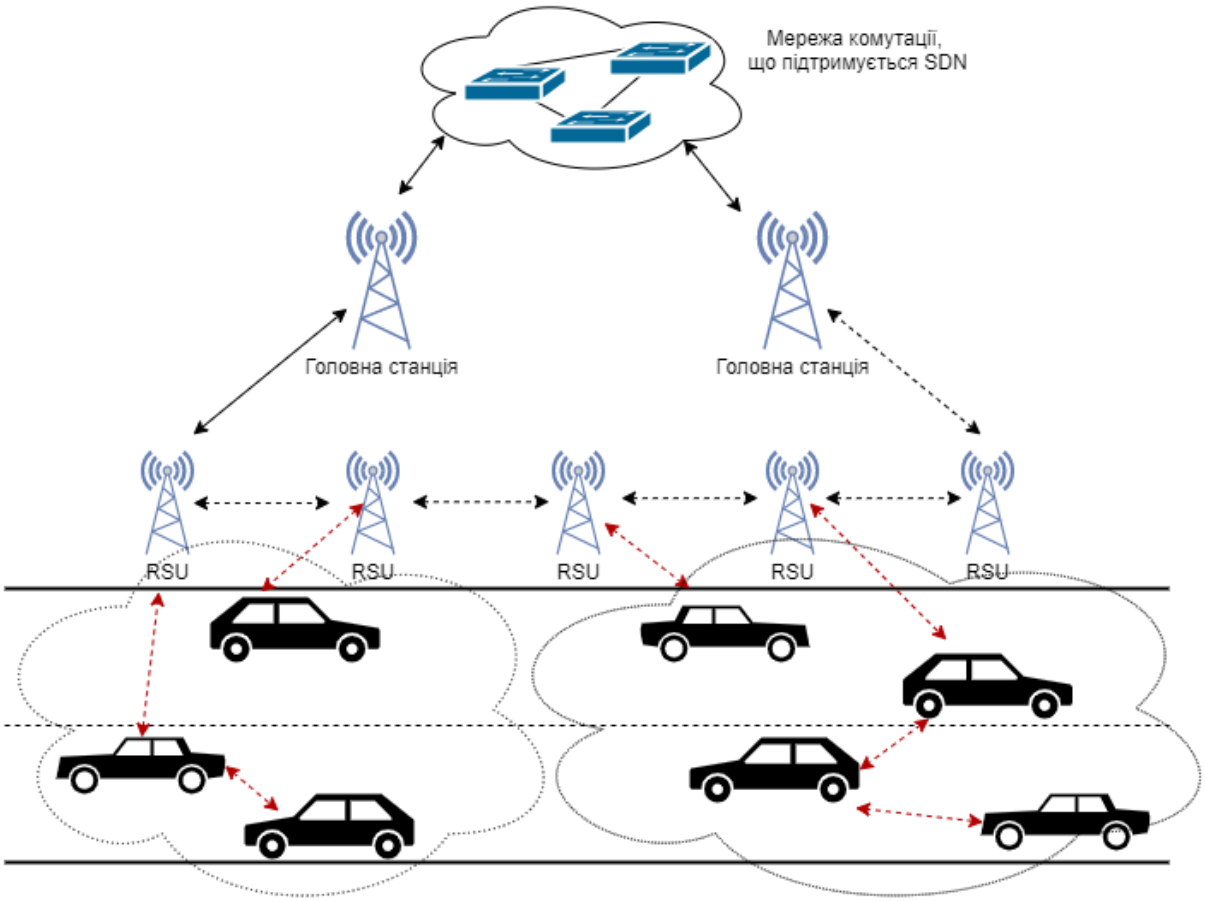


Рис. 1.5. Архітектура технології SDVN

1.4.1 Існуючі способи маршрутизації в SDVN

Важливими факторами використання технології SDN є точність алгоритму формування маршрутизації, надійність і ефективність передачі пакетів даних до відповідних вузлів. Оскільки це відносно нова область, відповідно, на даний момент, не існує великої кількості рішень відносно оптимального протоколу маршрутизації.

Один із прикладів реалізації SDVN наведено у роботі [13], у якій контролер бере на себе повну відповідальність за визначення вектору шляху. У наслідок такого алгоритму, отримавши запит на маршрут між вузлами, контролер визначає шлях від стартового вузла до вузла призначення,

застосовуючи популярний алгоритм пошуку найкоротшого шляху: алгоритм Діжкстри. При цьому метрикою шляху виступає мінімальний оптимістичний час, тобто час, за який потрібно зконектити два транспортні засоби, які рухаються у максимально вигідному, по відношенню один до одного, напрямі. На жаль, таке рішення не передбачує контролювання та врахування часу проходження, кількості необхідних для передачі пакетів, тощо, що залишає більшість описаних вище проблем маршрутизації в транспортних мережах все ще не вирішеними.

На відміну від попередньої системи, у роботі [14] контролер не має абсолютного контролю над конструюванням трафіку, а більше зорієнтований на керування вузлами, для вибору наступних кращих носіїв. У даній схемі маршрутизації автори реалізовують схему поширення та формування пакетів маршрутизації при передічі даних від стартової вершини до пункту призначення. На етапі поширення пакетів даних, стартовий вузол генерую множину копій поширюваних даних для пересилки іншим вузлам. Визначення вузлів, на які буде пересилатись нова інформація, визначаються за допомогою контролера орієнтуючись на кількість проміжних каналів між стартовою вершиною, та вершиною “пунктом призначення”. Суть етапу формування пакетів даних полягає у перерозподілі копій цих пакетів. Тут важливо враховувати напрямок переміщення кожного з вузлів, в напрямі від початкового до кінцевого. При застосуванні такого підходу вибір наступного вузла буде визначатись орієнтуючись не лише на відстань між вузлами, а і на суму кутів, що створює точка призначення. Ця процедура повторюється ітеративно, доки хоча б одна копія пакету даних не досягне шуканої вершини, Можна підкреслити, що подібна схема маршрутизації являє собою своєрідне поєднання VANET та SDVN оскільки застосовує стандартні функції обох технологій. Тим не менш, участь контролера на кожному етапі маршрутизації все ще створює ризик великих затримок та економічних витрат.

ВИСНОВОК ДО РОЗДІЛУ 1

Широка потреба у транспортуванні, та стрімкий розвиток прогресу у сучасних технологіях роблять використання транспортних засобів обов'язковою умовою теперішнього світу. Разом із збільшенням потреб, збільшується і кількість транспортних засобів, проте, площа дорожньої інфраструктури залишається незмінною. У результаті на сьогодні транспортна індустрія стикається з великою кількістю різноманітних проблем. Серед них: зростання тенденцій до аварій під час руху транспорту, довготривалі переміщення, систематична загруженість трафіку, пошкодження суспільної власності і у результаті ризик загрози життю людей.

Для вирішення існуючих проблем, з'явилася нова парадигма спеціальних транспортних мереж (VANET). Мережа VANET, у свою, чергу має певні особливості, які роблять існуючі протоколи маршрутизації не оптимальними для існуючої мережі. Серед таких унікальних характеристик можна виділити передбачуваність вектору руху, швидка змінюваність відстані між транспортними засобами та RSU, велика загруженість мережі, часті переключення, висока швидкість зміни топології та жорсткі обмеження на рахунок затримки.

Враховуючи той факт, що дослідження VANET не є новим, більшість існуючих проблем все ще не знайшли свого рішення. Наприклад, поширення неефективної інформації, проблеми безпеки та конфіденційності, незбалансований трафік, нестабільне використання ресурсів, відсутність якості, тощо [15].

На даний момент, популярним напрямом вирішення подібних проблем є включення технології програмно-визначених мереж у VANET. На початку розробки та дослідження, SDN в першу чергу було зорієнтоване на дротових мережах, особливо центрах обробки даних. Але на даний момент технологія SDN набуває все більшого поширення у бездротових та спеціальних доменах

мережах. Реалізація SDN у VANET має різноманітні варіації. Деякі дослідники пропонують керувати всією мережею за допомогою SDN, а у той час, як інші вважають, застосування SDN на магістральній мережі буде більш ефективними.

РОЗДІЛ 2. КОНСТРУЮВАННЯ ТРАФІКУ В ПРОГРАМНО ВИЗНАЧЕНИХ ТАНСПОРТНИХ МЕРЕЖАХ

2.1 Спосіб представлення транспортної мережі та її аналіз

Оптимальним способом візуалізації мережі є її представлення у вигляді графу. Оскільки ми маємо діло з транспортними мережами, правильним рішенням буде обрати реалізації навантажений граф $G = (V, E, W)$. Враховуючи особливості транспортної мережі, множина вершин графа $V = \{v_i \mid i = 1, 2, \dots, n\}$ позначає множину транспортних засобів у системі; множина граней графа $E = \{e_i\, j \mid j = 1, 2, \dots, k\}$ відображає множину відстаней між транспортними засобами; множина ваг ребер графу $W = \{w_j \mid j = 1, 2, \dots, m\}$ є умовним позначенням метрики шляху. У даному випадку, у якості метрики, ми використовуємо середній час надсилання інформації від одного транспортного засобу до іншого.

Відображення транспортної мережі у вигляді графа буде проводитись автоматично, і , відповідно, заплутаність та складність цього графу буде напряму залежати від транспортної розв'язки. Таким чином, граф маршрутів, що буде згенерований на автомагістралі, з транспортним кільцем та певною кількістю розгалужень, буде суттєво відрізнятись від того, що буде згенеровано на звичайному перехресті. Також варто звернути увагу на швидкість змінюваності усієї системи. Наприклад, таке явище як автомобільна "пробка", буде найстабільнішим та найнадійнішим варіантом для системи, у той час, як текучість транспорту вимагатиме частішого оновлення інформації про систему. Ще одним важливим фактором є умова відсутності транспортних засобів як таких, якщо концентрація транспорту буде надто низькою, то стабільна передача інформації, може бути просто неможлива. Беручи до уваги подібні ситуації та обмеження мережі, можна зробити висновок, що протокол маршрутизації повинен враховувати усі можливі варіанти розташування транспортних засобів та бути легко масштабованим.

У даній роботі для експерименту пропонується довільний граф з умовною кількістю транспортних засобів та зв’язків між ними. Структурну схему мережі можна побачити на рис. 2.1.

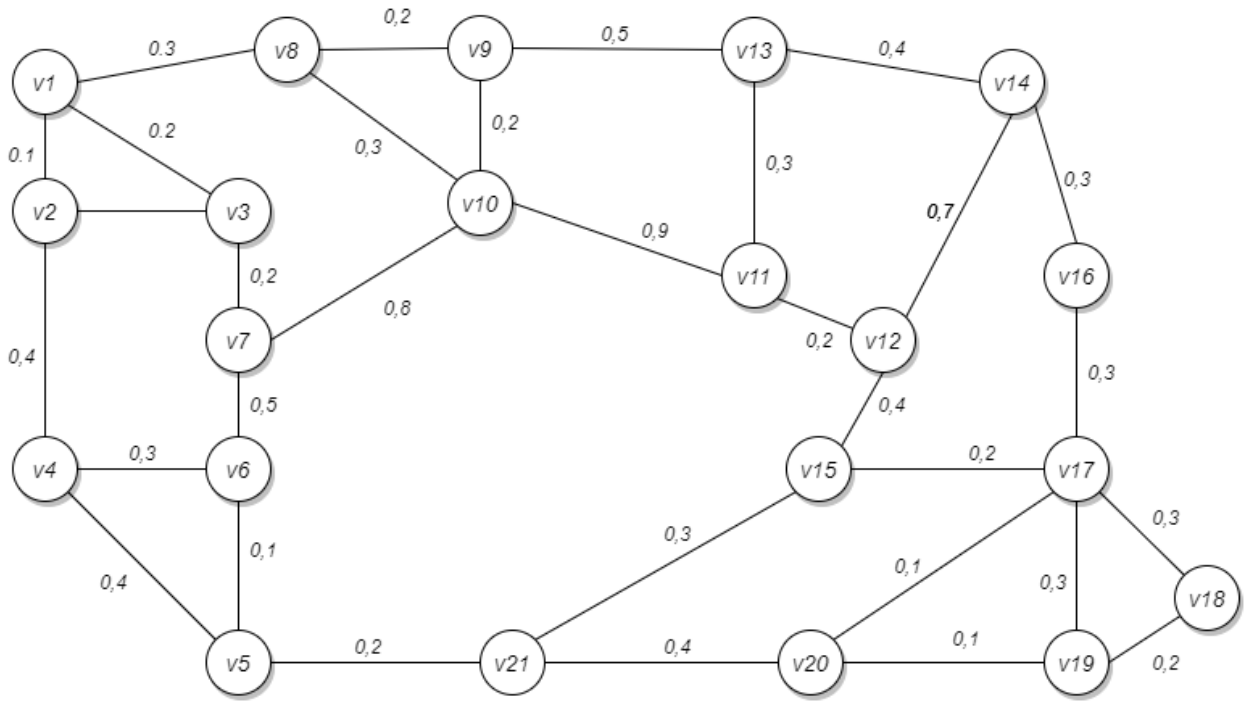


Рис. 2.1. Структурна схема транспортної мережі

Аналізуючи даний граф, слід також враховувати, що встановлена метрика може не співпадати з реальним масштабом відстаней між вершинами. Таке рішення обумовлене потребою тестування графа у найменш вигідних для нього умовах. Таким чином, встановлюються значення за яких алгоритм, теоретично, я найбільш уразливим, при цьому заданий граф зберігатиме візуальну сприйнятність.

Маючи велику кількість розгалужень, а отже, і велику кількість непересічних шляхів, заданий граф виступає прикладом однієї із складних систем, коли існує можливість встановлення відносно великої кількості зв’язків.

Для прикладу, у ході експерименту буде визначатися шлях для двох найвіддаленіших вершин. У даному випадку, вихідною вершиною буде

вершина з першим порядковим номером, а кінцевою вершина з номером шістнадцять.

2.2 Алгоритм формування маршрутної інформації

У процесі конструювання трафіку, для кожного транспортного засобу, у контролері SDN створено окремий віртуальний комутатор з індексом, що відповідає номеру вершини га графі. Головним інструментом в формуванні трафіку буде фігурувати таблиця маршрутів віртуального комутатора. Така таблиця, зберігає дані про маршрути, що проходять через дану вершину. Серед них: ідентифікатор контролера, пункт призначення, суміжна вершина, що встановлює вектор шляху у напрямі до кінцевого вузла, шлях від зазначеного до кінцевого вузла, метрика цього шляху та його завантаженість. Як правило, за необхідності отримання більш повного аналізу системи, цілком реальним є додавання нової актуальної інформації. У свою чергу, для реалізації запропонованого алгоритму, наведеного стеку даних достатньо.

Алгоритм формування маршрутної інформації передбачає створення записів в таких таблицях маршрутизації для кожного з непересічних шляхів від початкового до кінцевого вузла.

У своїй роботі я пропоную варіацію хвильового алгоритму маршрутизації. Формування маршрутної інформації реалізовується у зворотному напрямі, від кінцевого до стартового вузла. Перевага такого способу полягає у забезпеченні можливості генерування множини непересічних шляхів, збільшуючи свою віддаленість від точки призначення. У ході роботи такого алгоритму, також формується множина шляхів від проміжних вершин до кінцевого, завдяки чому зникає необхідність повторного запуску алгоритму від проміжних вершин.

Конструювання трафіку відбувається за рахунок послідовного виділення суміжної множини вершин V_i та V_{i+1}. Поняття суміжної множини розуміються множини $V_i = \{v_i \mid i = 1, 2, \dots, n\}$ та $V_{i+1} = \{v_i \mid i = 1, 2, \dots, n\}$

сполучені умовною множиною ребер $E_i = \{e_i\, j \mid j = 1, 2, \ldots, k\}$, у яких $v_k \in V_i$ та $v_m \in V_{i+1}$. Оскільки формування маршрутної інформації починається з кінцевої вершини графу, що відповідає віртуальному комутатору S_i, то перша отримана множина буде визначена, як $V_i = \{v_{18}\}$, а суміжна до неї визначатиметься як $V_2 = \{v_{19},\ v_{17}\}$. На рисунку 2.2. новоутворені множини виділено пунктиром.

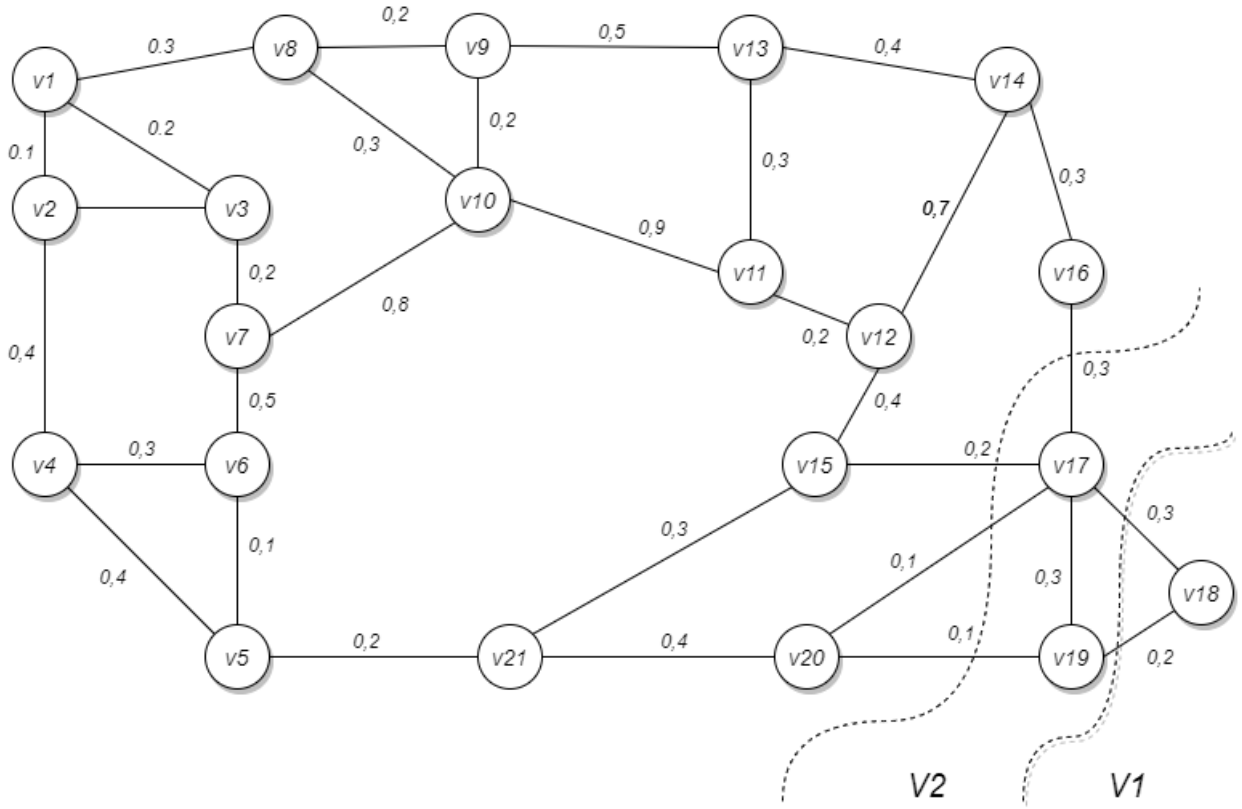


Рис. 2.2. Результат виділення першої та суміжної до неї множин

Таким чином на етапі першої ітерації ми отримуємо по два шляхи від суміжних вершин, починаючи з вершини номер сімнадцять: 1) $v_{17} \rightarrow v_{18}$, 2) $v_{17} \rightarrow v_{19} \rightarrow v_{18}$, та для дев'ятнадцятої вершини графу 1) $v_{19} \rightarrow v_{18}$, 2) $v_{19} \rightarrow v_{17} \rightarrow v_{18}$.

Для кожного із знайдених шляхів, у таблиці маршрутизації комутатора буде створено відповідний запис. Ідентифікатор комутатора збігається з порядковим номером вершини, що служить початком знайденого шляху. Суміжна вершина, що визначає вектор шляху, як правило, друга вершина

першого каналу зв'язку. У нашому випадку, для двох перших шляхів сімнадцятої вершини суміжними будуть v_{18} та v_{19} відповідно, а для дев'ятнадцятої v_{18} та v_{17} відповідно. Метрика всього шляху визначається методом сумування ваг каналів зв'язку, що складають цей шлях. Також для кожного із шляхів буде визначено та записано показник навантаженості. Таким чином проходить формування маршрутної інформації в таблиці віртуального комутатора. Даний процес буде повторюватись до тих пір, поки стартова вершина не буде досягнута у новоствореній множині суміжних вершин. У результаті ми отримуємо наступне розподілення системи, що зображено на рисунку 2.3.

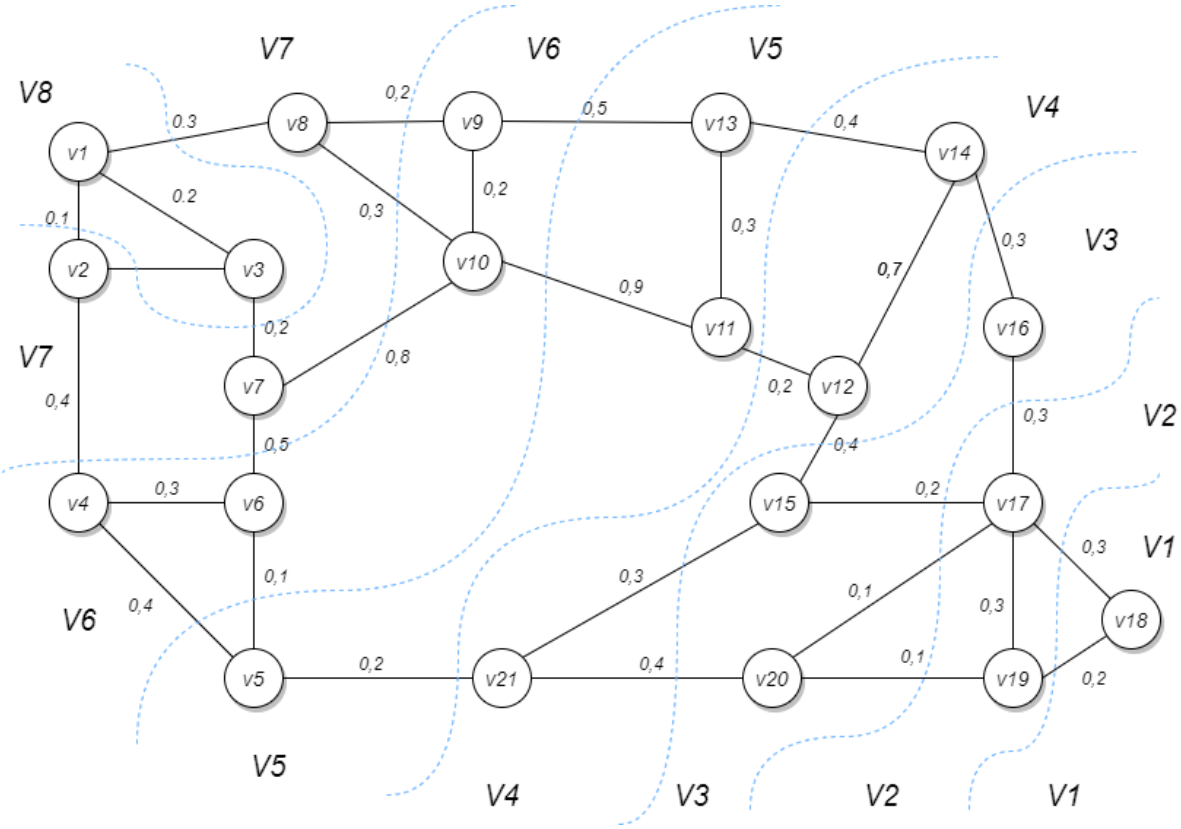


Рис. 2.3. Результат розподілення мережі на множини суміжних транспортних засобів

Результатом систематичного процесу обміну маршрутної інформації між комутаторами одного та суміжного рівня є формування нових записів у таблиці маршрутизації заданих вузлів графа транспортної мережі. Кількість

записів в таблиці буде відповідати кількості непересічних шляхів від стартової чи проміжної вершини до кінцевої.

Таблиця 2.1 відображає маршрутну інформацію стартової вершини, і зберігає дані про оптимальний шлях до кінцевої вершини, у даному випадку — вісімнадцятої.

Таблиця 2.1. Таблиця маршрутизації контролера S_1

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
1	v18	v8	2.9	D_1	v1 → v8 → v10 → v11 → v12 → v15 → v17 → v18
1	v18	v3	1.9	D_2	v1 → v3 → v7 → v6 → v5 → v21 →v20 → v19 → v18
1	v18	v2	1,8	D_3	v1 → v2 → v4 → v6 → v5 → v21 →v20 → v19 → v18

На перший погляд, аналізуючи наведену таблицю, можна зробити висновок, шлях через суміжну вершину з порядковим номером два є найоптимальнішим. Таке рішення обумовлене найнижчим значенням метрики шляху у таблиці. Проте, слід також звернути увагу на показник загрузки шляху. На жаль ручний підрахунок загруженості системи є доволі трудомістким процесом і тому у таблиці теоретичних значень цей показник пропущений. Тим не менш, спробуємо розглянути всі можливі варіанти.

Отже, ми маємо три робочі сценарії. За першим сценарієм найнижчий показник загруженості системі збігається з найнижчим показником метрики шляху у таблиці маршрутизації. У такому випадку вибір оптимального шляху є більш ніж очевидним. За другим сценарієм, найнижчий показник загруженості належить шляху з суміжною вершиною з ідентифікатором три.

У такому випадку, даний шлях буде визначено, як найбільш оптимальний. Таке рішення обумовлене тим, що метрика у двох шляхів приблизно рівна, але коефіцієнт загруженості другого шляху у таблиці нижчий. За третім сценарієм ми маємо кардинально різні показники загруженості та метрики шляху. При виникненні такої проблеми, коли постає вибір надати перевагу тому чи іншому показнику, варто приймати рішення беручи до уваги стан усієї мережі. Якщо на даний момент щільність розміщення транспортних засобів не є високою, то звичайно, шлях з найменшою довжиною шляху буде вважатись найоптимальнішим. За умови відсутності значень загруженості шляхів, найкоротший шлях буде обиратись, орієнтуючись на значення метрики. На рисунку 2.4 цей шлях виділено синім кольором.

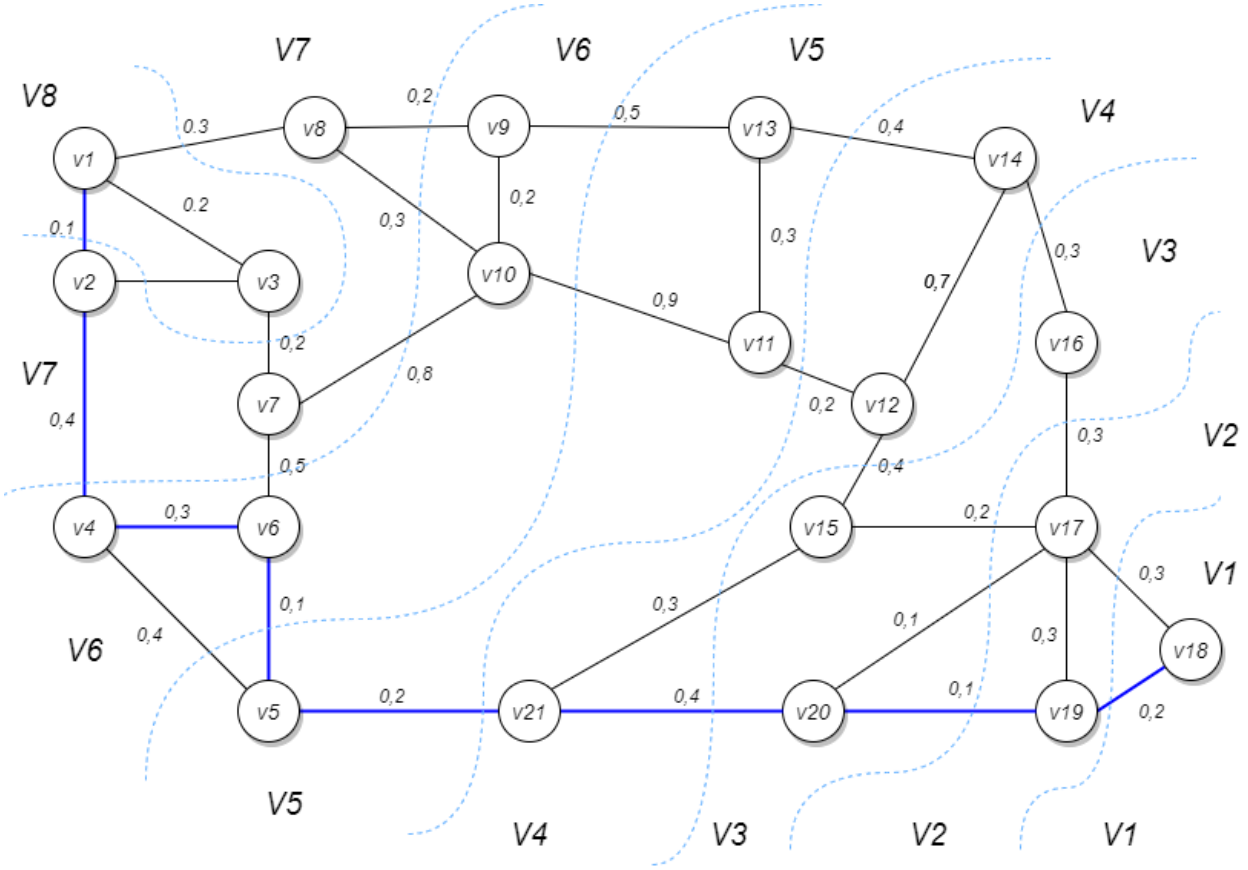


Рис. 2.4. Результат роботи алгоритму формування маршрутної інформації

Функціональну схему алгоритму наведено на рисунку 2.4.

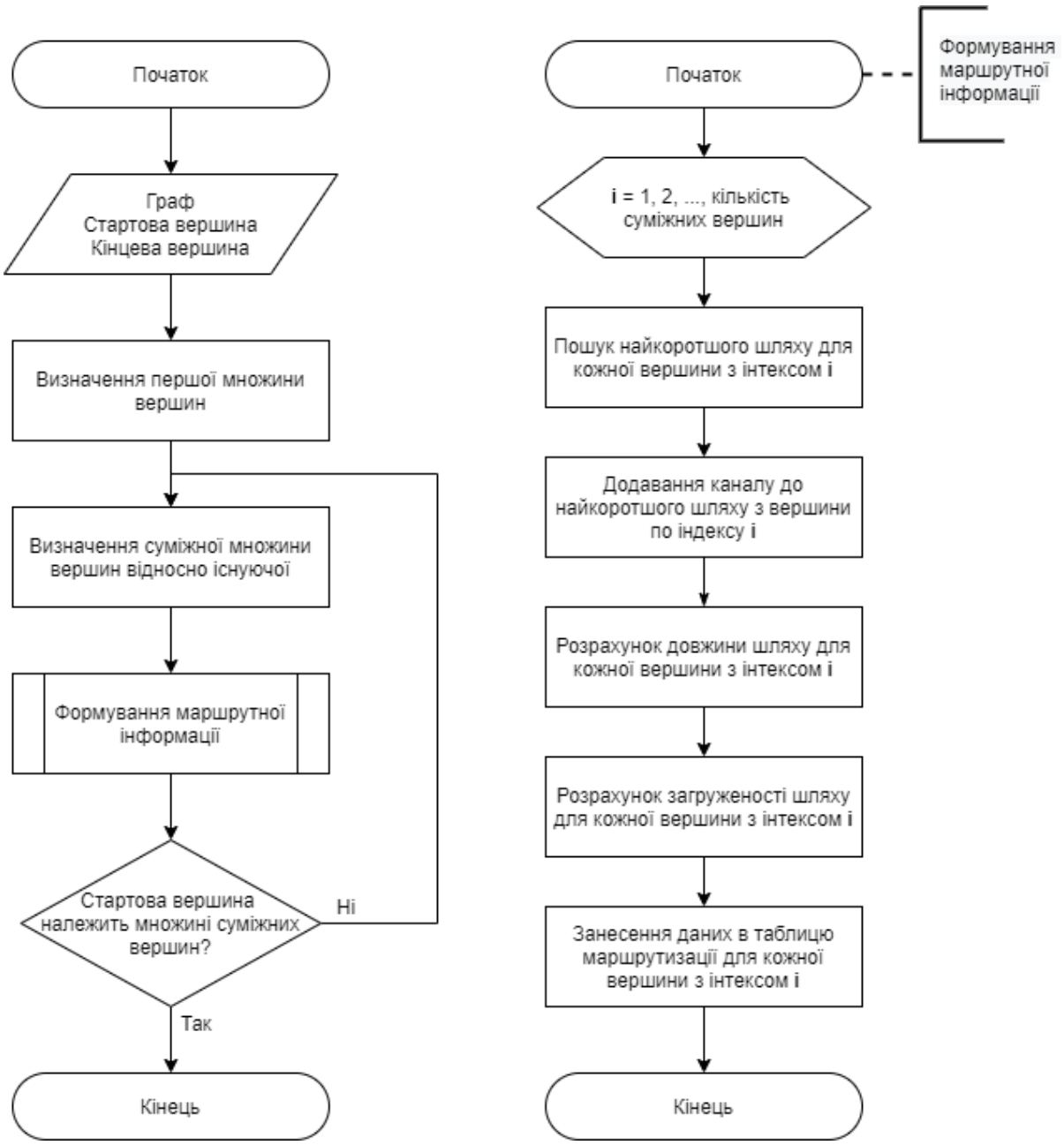


Рис. 2.5. Функціональний алгоритм формування маршрутної інформації

2.3 Формування балансування трафіку

Одним із головних недоліків існуючих методів курування маршрутизацією є перенаправлення великої кількості потоків даних на один канал зв'язку. Як правило, це обумовлено тим, що найкоротший шлях є найоптимальнішим і, відповідно, найбільш часто використовуваним. Подібна проблема може сприяти утворенню дисбалансу трафіку мережі та втраті полоси пропускання [16]. При конструюванні трафіку, та його балансуванні

необхідно враховувати метрику та навантаження каналів зв’язку. Такий процес є найбільш актуальним саме для транспортних мереж, у яких висока загруженість каналів призводить да утворення заторів на дорогах. Запропонована у даній роботі система не є виключенням.

Як вже було наголошено, швидкість руху транспортних засобів, а отже і час обміну інформацією напряму залежить від щільності розташування транспортних засобів. Відсутність стабільності по відношенню до щільності розташування вершин призводить також до зниження середніх показників швидкості руху транспортних засобів а також пропускної здатності усієї мережі. Найбільш оптимальна відстань залежить від квадрату швидкості руху транспорту [17]. Таким чином, за умови підвищення щільності розтушування транспортирних засобів на дорогах, відповідно буде знижуватись швидкість переміщення вузлів та пропускна здатність автомагістралей.

Беручи до уваги вище описані проблеми, можна припустити, що деякий шлях може бути вибраним як найоптимальніший, виключно за умови, що загруженість каналів зв’язку, буде мати не високі показники. Таким чином, при формуванні трафіку, будуть обиратись шляхи з мінімальною нагруженістю, та мінімальним середнім квадратичним відхиленням нагруженості шляхів від середнього значення. У роботі [17] запропоновано використання наступного критерію рівномірності загруженості каналів шляху:

$$D_i = (d_i^0 + \sum_{j=1}^{n} \frac{l_i(d_j - d_i^0)^2}{L_i}), \quad (2.1)$$

де n — кількість каналів заданого шляху від початкового вузла до кінцевого;

d_i^0 — середнє значення загруженості каналів заданого шляху;

d_i — загруженість каналу шляху;

L_i — довжина заданого шляху;

l_j — довжина каналу заданого шляху.

Середнє квадратичне відхилення загруженості каналів заданого шляху ($(d_j - d_i^0)^2$) характеризує ступінь рівномірності загрузки каналів шляху. Значення величини d_i^0 визначає найменший показник загруженості шляху. Для його обрахунку, у роботі [17] запропонована наступна формула:

$$d_i^0 = \frac{1}{n} \sum_{j=1}^{n} \frac{l_j \cdot d_j}{L} \tag{2.2}$$

Вибір оптимального шляху з урахуванням описаного критерію сприяє рівномірному розподіленню загруженості заданої транспортної мережі. Таким чином, показник, загруженості, що зберігається у таблиці маршрутизацій SDN контролерів, є важливим інструментом оптимізації конструювання трафіку.

2.4 Алгоритм конструювання трафіку в програмно визначених транспортних мережах

Існує декілька переваг використання заданого алгоритму маршрутизації. У першу чергу слід враховувати той факт, що транспортні засоби можуть рухатись дуже нестабільно та швидко, постійно змінюючи напрям свого руху. Отже, формування нового шляху має проходити у максимально швидкому режимі.

Особливістю запропонованого способу конструювання трафіку є збереження інформації маршрутизації у базі даних. Таким чином, чим більше разів застосовується алгоритм формування множини шляхів, тим більше маршрутної інформації про мережу ми отримуємо і, відповідно, тим менше виникає необхідність запуску цього алгоритму повторно.

Розглянемо гіпотетичну ситуацію на фоні проведеного теоретичного аналізу у розділі 2.2. Конструювання трафіку транспортної мережі буде відбуватись згідно функціональної схеми, зображеної на рисунку 2.6.

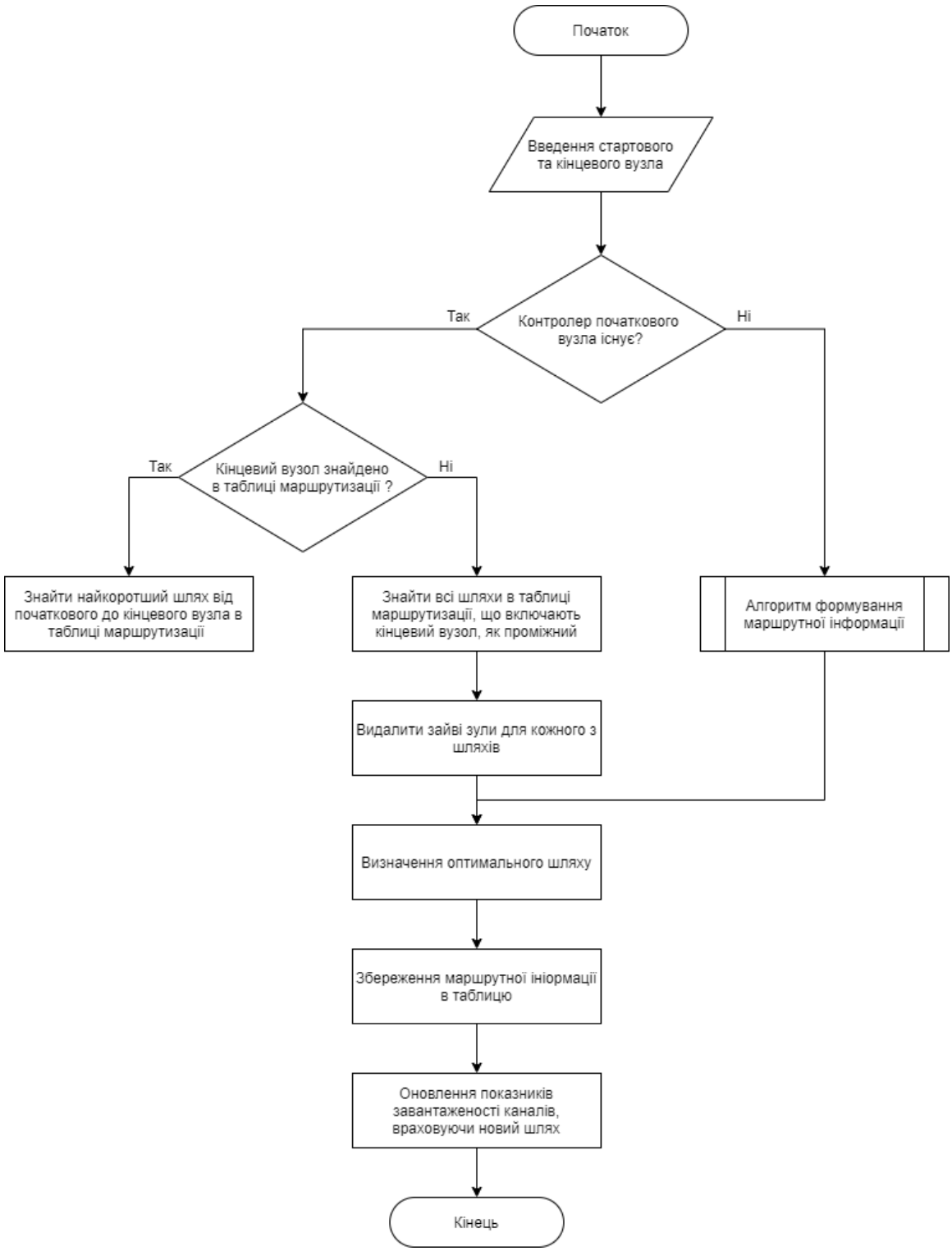


Рис. 2.6. Функціональна схема алгоритму конструювання трафіку в транспортної мережі

Припустимо, що у нас виникає необхідність конструювання маршруту від десятої до п’ятнадцятої вершини, паралельно по відношенню попередньо

знайденого шляху. У результаті ми маємо подальший розвиток подій: оскільки мінімальна кількість каналів між стартовою та кінцевими вершинами першого шляху більша за мінімальну кількість каналів другого шляху, і стартова вершина другого шляху є проміжною вершиною до першого, можна зробити припущення, що при паралельному визначенні двох маршрутів цілком можливим є формування двох маршрутів за один акт роботи алгоритму. Шуканий другий маршрут може бути отриманим як проміжний з бази даних.

Подібний алгоритм демонструє розвиток трьох сценаріїв розвитку подій. Якщо стартова вершина нового шуканого шляху є проміжною вершиною першого, то для нього буде існувати контролер з актуальною таблицею маршрутизації. У такому випадку постає питання, чи є хоча б один шлях, у якому наявна кінцева вершина шуканого шляху. Можливий випадок, що такий шлях буде повність зберігатись в таблиці. У такому випадку, залишається просто дістати потрібний шлях з бази даних.

Дещо складнішою є ситуація, коли кінцевий вузол, наявний лише як проміжна частина одного з шляхів. У такому випадку, необхідно з усіх шляхів, що мають у своєму складі кінцевий вузол, видалити зайві канали та визначити показники загруженості та метрики шляху. Відповідно до цих показників, заповнити таблицю маршрутизації. У випадку, що контролер шуканої вершини не знайдено в базі даних, необхідно запустити алгоритм формування маршрутної інформації. Об’єднуючим фактором для цих сценаріїв є обов’язкове оновлення показників загруженості каналів, з урахуванням кількості робочих маршрутів транспортної системи.

Розглянемо другий сценарій на прикладі існуючої таблиці маршрутизації контролера восьмої вершини заданого графу. Нехай нам необхідно знайти оптимальний маршрут від восьмої вершини до п’ятнадцятої. У таблиці 2.2 наведено дані таблиці маршрутизації восьмого контролера.

Таблиця 2.2. Таблиця маршрутизації контролера S_8

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
1	v8	v9	2.9	D_1	v8 → v9 → v13 → v11 → v12 → v15 → v17 → v18
1	v8	v10	1.9	D_2	v8 → v10 → v11 → v12 → v15 → v17 → v18

Аналізуючи задану таблицю, можна побачити, що шуканий шлях двічі зберігається у таблиці маршрутизації, отже наступним кроком формування маршруту є виділення цих шляхів та подальше видалення зайвих каналів. У такому випадку ми отримаємо шляхи: 1) v8 → v9 → v13 → v11 → v12 → v15; 2) v8 → v10 → v11 → v12 → v15; Визначення найоптимальнішого шляху залежить від показників загруженості шляху та метрики шляху. Показник загруженості каналів зв'язку не буде враховуватись під час теоретичного розрахунку, через складність обрахунку та відсутності можливості симуляції цілої системи. Під час проведення програмного експерименту, буде запущено механізм симуляції системи та обрахунок загруженості каналів зв'язку.

Таким чином головним фактором вибору виступатиме саме метрика шляху. На основі таблиці 2.2, можна зробити висновок, що перший шлях є більш оптимальним, оскільки його метрика дорівнює 1.6 , у той час як метрика другого шляху дорівнює 1.8. У такому випадку, у таблиці маршрутизації восьмого вузла буде створено новий запис про шуканий шлях. Очевидно, що це призведе до певних змін і в інших таблицях маршрутизації, адже з появою нового маршруту, загруженість деяких каналів зросте. Оновлення інформації загруженості є обов'язковим, оскільки додавання нових шляхів напряму впливає на швидкість та надійність передачі даних.

Результат редагування та оновлення маршрутних даних можна побачити у таблиці 2.3.

Таблиця 2.3. Таблиця маршрутизації контролера S_8 після оновлення

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
1	v8	v9	2.9	D_1	v8 → v9 → v13 → v11 → v12 → v15 → v17 → v18
1	v8	v10	1.9	D_2	v8 → v10 → v11 → v12 → v15 → v17 → v18
1	v8	v9	1.6	D_3	v8 → v9 → v13 → v11 → v12 → v15

Таким чином, на рисунку 2.7 зображено граф транспортної системи, з двома новими шляхами, що підсвічені синім.

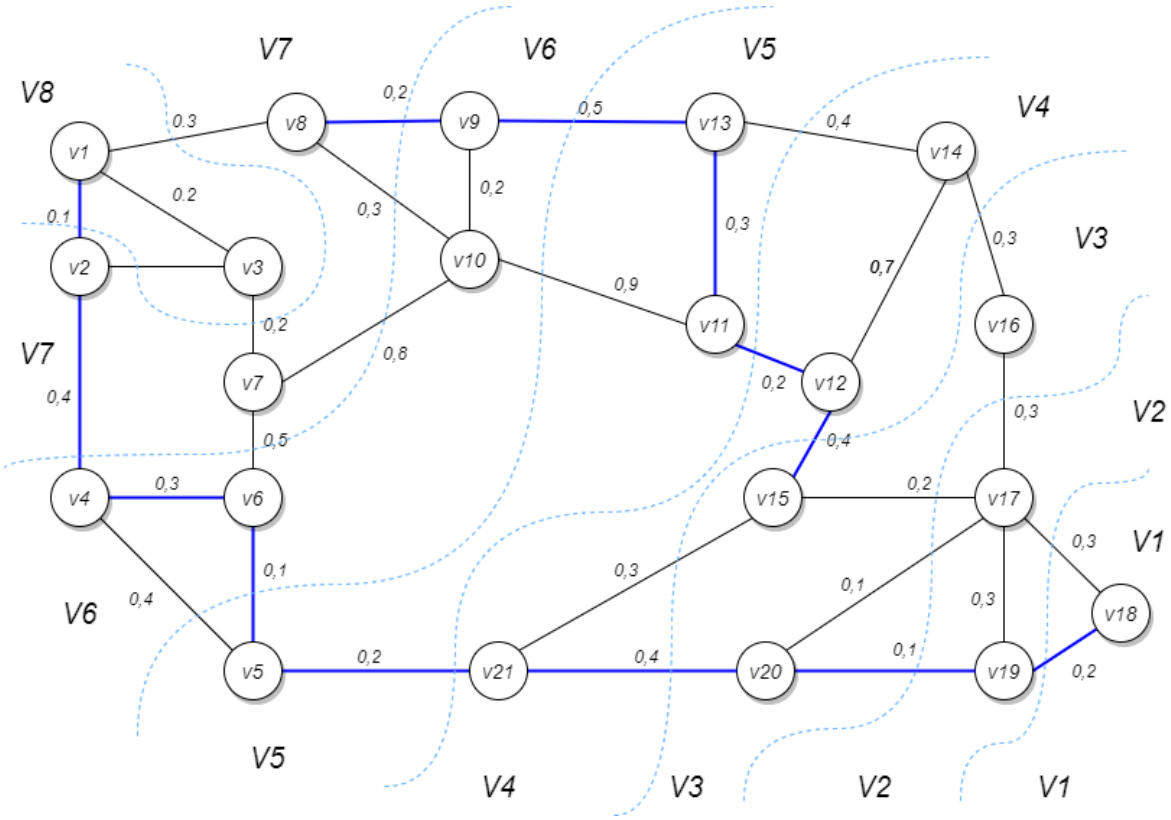


Рис. 2.7. Результат роботи алгоритму конструювання трафіку в транспортної мережі

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було запропоновано модифікацію алгоритму формування маршрутної інформації транспортної мережі. Оскільки передбачити вигляд системи неможливо в силу різноманітності доріг, різної швидкості руху транспортних засобів та щільності їх розташування, то найбільш зручним способом представлення такої інтелектуальної мережі є представлення у вигляді графа.

На базі запропонованого графа було застосовано алгоритм маршрутизації для пошуку оптимального маршруту між двома найбільш віддаленими вершинами. У нашому випадку – це перша та шістнадцята вершини. У ході експерименту отримано та записано дані у таблиці маршрутизації цих вершин та усіх проміжних вершин системи. Також для проведення повного аналізу системи, також наведено спосіб розрахунку загруженості каналів та шляхів системи, методом балансування трафіку.

Результатом роботи такого алгоритму є граф на рисунку 2.4 з виділеним шуканим шляхом та таблиця маршрутизації контролера першого вузла. Також було проведено теоретичний експеримент алгоритму конструювання трафіку з застосуванням технології попереднього аналізу маршрутної інформації в базі даних.

На рисунку 2.7 зображено граф з двома шуканими шляхами, що були визначені як найоптимальніші за даних умов у даній транспортній системі. Визначення обох шляхів було проведено за один акт роботи алгоритму. У таблиці 2.3 наведено результат обчислення маршрутної інформації другого шляху.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Використані технології розробки програмного забезпечення

В даній роботі буде проведено проектування та розробка програмного забезпечення для реалізації алгоритму конструювання трафіку в інтелектуальних транспортних мережах.

3.1.1 Обумовленість вибору мови програмування

Моделювання запропонованого алгоритму буде проводитись на мові програмування Java. Такий вибір обумовлений рядом наступних чинників:

- *Java є об'єктно-орієнтованою мовою програмування*

Дана характеристика робить обрану мову надзвичайно популярною серед розробників програмного забезпечення. Отримавши знання про такі ключові поняття як абстракція, інкапсуляція, поліморфізм та наслідування ви можете зберегти модульну, гнучку та розширювану систему. Java також сприяє використанню принципів проектування, орієнтованих на SOLID та об'єкти, у вигляді проектів з відкритим кодом, таких як Spring, які гарантують, що ваша об'єктна залежність добре керується за допомогою принципу Dependency Injection.

- *У Java є багатий API*

Java надає API для вводу/виводу, мереж, утиліт, розбору XML, підключення до бази даних та майже все, що необхідно для продуктивної розробки. Для задоволення потреб розробника існує безліч бібліотек з відкритим кодом, таких як Apache Commons, Google Guava, Jackson, Gson, POI Apache тощо.

- *Наявність розвинених інструментів для розробки*

Eclipse, Netbeans та IntelliJ IDEA надають потужну можливість налагодження. Інтеграційне середовище розробки, або (IDE) створює можливість використання більшої кількості інструментів для розробників на

Java. Серед часто використовуваних інструментів також існує Maven, ANT, Jenkins, декомпілятори, JConsole, Visual VM.

- *Широкий стек бібліотек*

Подібна характеристика є важливою причиною, чому Java стала настільки популярною та використовується майже всюди. Розробниками нових бібліотек є такі масштабні організації як Google, Oracle, Apache і прості розробники, що бажають покращити існуючі рішення.

- *Розвинута підтримка Community*

Існує безліч активних форумів (StackOverflow, GitHub) та організацій з відкритим кодом користувачів Java, які допомагають іншим розробникам. Багато програмістів, які використовують відкритий код, роблять внесок у загальну розробку продукту.

- *Java є безкоштовною мовою програмування*

Доступність мови програмування та її безкоштовна підтримка ще одна важлива річ, яка змушує організацію вибирати Java для свого стратегічного розвитку.

- *Належна документація*

Javadoc полегшив розробку програмного забезпечення та забезпечив додаткові можливості аналізу та вдосконалення коду під програмування.

- *Java є кросплатформною мовою програмування*

Це все ще є однією з причин того, що Java є найкращою мовою програмування, оскільки програма розроблена на одній операційній системі може безпроблемно працювати на будь-якій іншій платформі.

3.1.2 Обумовленість вибору фреймворку Spring для розробки

Spring - це потужний фреймворк, що використовується для розробки додатків. У більш широкому розумінні можна сказати, що Spring Framework - це чітко визначений інструмент, який підтримує декілька веб-додатків, що

використовують Java як мову програмування. Найважливіші Spring Framework:

1) *Spring є дуже легким відносно своїх розмірів та функціональності*. Це пов’язано з його реалізацією POJO, яка не примушує його успадковувати будь-який клас або реалізовувати будь-які інтерфейси.
2) *Аспект-орієнтоване програмування (AOP)* є важливою частиною Spring. Програмування, орієнтоване на аспекти, використовується для відокремлення наскрізних проблем (логінг, безпека, тощо) від ділової логіки програми.
3) *Управління транзакціями* використовується для координації транзакцій для об’єктів Java. Крім того, він не прив'язаний до середовища J2EE і використовується з безконтейнерними середовищами.
4) *Контейнер*. Структура Spring розробляє та керує життєвим циклом та конфігураціями об’єктів додатків.
5) *Ін'єкційна залежність (Dependency Injection)* є особливістю Spring Framework, яка дозволяє розробляти не сильно пов'язані програми. Таким чином, тестування цих слабко пов'язаних додатків стає спрощеним, а також дозволяє розробнику змінити певні модулі відповідно до своїх потреб.
6) Інтеграція з іншими фреймворками, такими як IBATIS, Hibernate, Toplink тощо.

3.1.3 Обумовленість вибору бази даних PostgreSQL

Надзвичайно важливим фактором розробки програмного забезпечення є реалізація продукту правильним чином в плані паралелізму. Таке рішення є універсальним з точки зору побудови архітектури (CRUD, OLTP, OLAP, Analytics) і потужним з точки зору обчислення.

PostgreSQL - це найсучасніша система управління базами даних загального призначення з відкритим кодом. Незважаючи на те, що розробка цієї технології була призначена для використання на Unix платформі,

PostgreSQL був розроблений як кросплатформний, що забезпечує стабільну роботу на різних платформах, таких як Mac OS X, Solaris та Windows.

Важливою перевагою використання запропонованої системи управління базами даних також є безкоштовне програмне забезпечення з відкритим кодом. Ви можете безкоштовно використовувати, змінювати та поширювати PostgreSQL у будь-якій формі, використовуючи ліберальну ліцензію.

3.2 Модель та структура системи

Дана система складається з сервісної частини, об'єктної частини та користувацького інтерфуйсу.

3.2.1 Основні сервіси аналізу системи

Для конструювання трафіку було реалізовано три основні сервіси, такі як : *TrafficEngineeringService, RoutingAnalisisService,* та *LoadingAnalisisService*.

1) Сервіс *TrafficEngineeringService*

Призначений для конструювання трафіку транспортної мережі та є прямою реалізацією алгоритму, що зображено на рисунку 2.6. Даний сервіс використовує подані на вхід дані для початку розрахунку та видає остаточний результат. Його особливістю є пряме використання алгоритму збору маршрутної інформації та прив'язка до інших сервісів системи. За допомогою цього сервісу також було проведено детальний аналіз мережі та збір таких даних як час роботи алгоритму та часова складність алгоритму.

Основним методом є *startTrafficEngineering(Vertex source, Vertex destination)*, на вхід якого подаються стартова та кінцева вершини. Результат роботи даного методу, оптимальний маршрут по заним вершинам, буде виведено на спеціальній панелі користувацького інтерфейсу та у консолі додатку. Разом з шуканим шляхом також виводиться його повна аналітика, така як загруженість мережі час виконання та часова складність.

2) Сервіс *RoutingAnalisisService*

Відповідає безпосередньо за збір та аналіз маршрутної інформації системи. Даний сервіс є реалізацією алгоритму формування маршрутної інформації, що відображено на рисунку 2.5. Під час виконання головного методу цього сервісу *performRoutingDataAnalisis* формується множина суміжних вершин, починаючи з кінцевої вершини, но приходить як параметр на вхід даного метода. Для зручності створено спеціальний об'єкт *AdjancyVertices,* поля якого зберігають інформацію про поточну та попередню множину вершин.

Для кожної вершини з множини буде створено необхідну кількість записів у таблиці маршрутизації відповідного контролера. Для цього спочатку визначається множина усіх можливих векторів руху у напрямі до початкової вершини. Далі ця множина відфільтровується шляхом відкидання дублюючих, пройдених та визначних вершин. У результаті для кожного вузла буде створено окремий запис у базі даних, що зберігає інформацію про контролер, вектор напряму руху, метрику цілого шляху, сам шлях, а також стартову та кінцеву вершину.

Під час тестування доводилось стикатись з проблемою, коли збір маршрутної інформації був не можливий через недостатню кількість інформації про систему. Для вирішення цієї проблеми використовується спеціальна черга. Таким чином, якщо для визначення та збереження певної маршрутної інформації для окремої вершини на даний момент є неможливим, ця вершина буде збережена в чергу і пройдена ще один раз після збору інформації для усіх вершин, що належать поточній множині суміжних вершин.

Послідовність цих дій буде продовжуватись поки не буде досягнуто стартового вузла, а в базі даних не буде збережено шуканого оптимального маршруту.

3) Сервіс *LoadingAnalisisServce*

Виконує аналіз існуючої моделі транспортної системи та має у своєму складі два основні методи: *performLoadingAnalisisServce()* та *calculateLoadingCreterion().*

Метод *performLoadingAnalisisServce()* здійснює аналіз загруженості для кожного із каналів транспортної системи, враховуючи множину активних шляхів системи та прогнозування майбутнього стану цієї системи.

Метод *calculateLoadingCreterion()* використовує формулу балансування трафіку (2.1) з попереднього розділу для визначення критерію загруженості для кожного шляху. Цей розрахунок у головним чином базується на попередньому аналізі загруженості усієї системи. Саме результат обчислень за даною формулою зберігається в базі даних для кожного з шуканих маршрутів як “завантаженість шляху”.

На рисунку 3.1 зображено UML діаграму описаних сервісів.

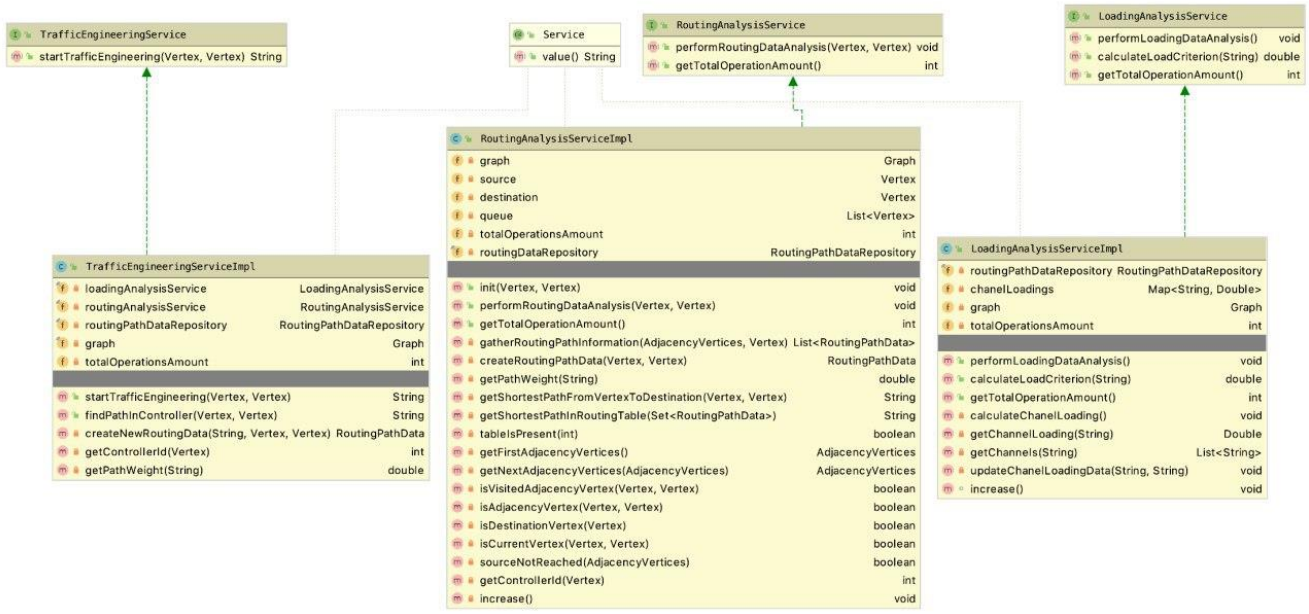


Рис. 3.1 UML діаграма сервісів

Дана діаграма відображає множину методів та полів для проведення аналізу транспортної системи базуючись на різних аспектах, таких як

завантаженість трафіку, формування маршрутної інформації та оцінка часової складності алгоритму.

3.2.2 Опис моделі системи

Основною моделлю транспортної системи є граф, програмно представлений екземпляром класу *Graph*. Даний клас має такі приватні поля як мапа *adjacencyVertices* (використовується для збереження суміжних вершин відносно до заданої) та *edgesWeights* (використовується для збереження вар ребер). Формування графа стає можливим завдяки таким методам керування як: *addVertex()* (додавання нової вершини), *removeVertex()* (видалення вершини), *addEdge()* (з'єднання двох вершин або встановлення ребра), *removeEdge()* (видалення ребра заданого графа). Для керування вузлами графа використовується клас *Vertex*, який зберігає у собі лише власну назву в якості ідентифікатора. На рисунку 3.2 зображено діаграму класів моделі системи.

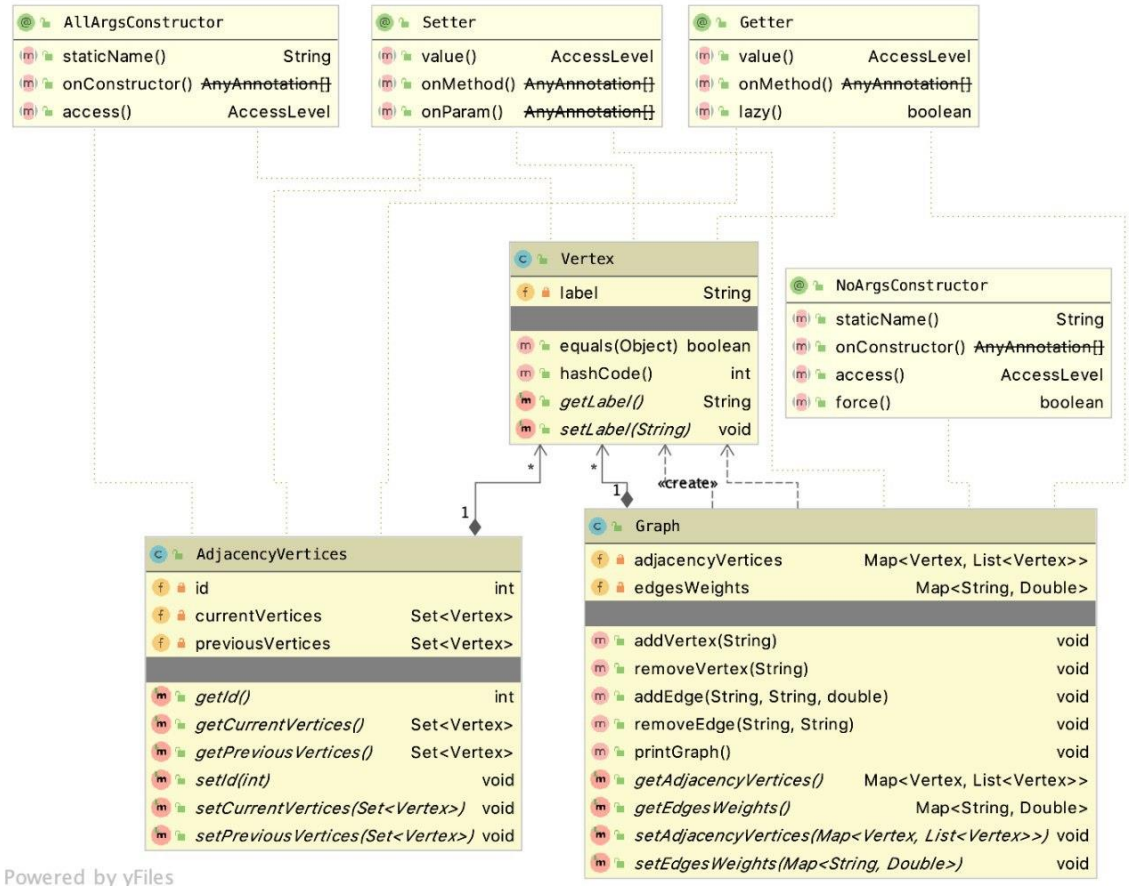


Рис. 3.2. UML діаграма моделі системи

На наведеній діаграмі можна помітити анотації *@AllArgsConstructor*, *@NoArgsConstructor*, *@Setter* та *@Getter*. Дані анотації належать бібліотеці *Lombok*, за допомогою яких у рантаймі генеруються методи встановлення та отримання значень полів конкретного класу.

3.2.2 Візуалізація створеної системи

Відображення заданого користувачем графа здійснюється завдяки розширенню відкритої бібліотеки *JFrame*. Взаємозв'язок між класами *Node*, *Edge* та *GraphDraw* влаштований подібним чином як програмна модель транспортної системи. Відмінність полягає у тому, що *GraphDraw* відповідає за встановлення вершин та ребер у графічному інтерфейсі. Також дадані методи для встановлення заготовлених графів без подальшого введення даних. Таким чином існує можливість встановлення власного розміру, форми та положення вершини.

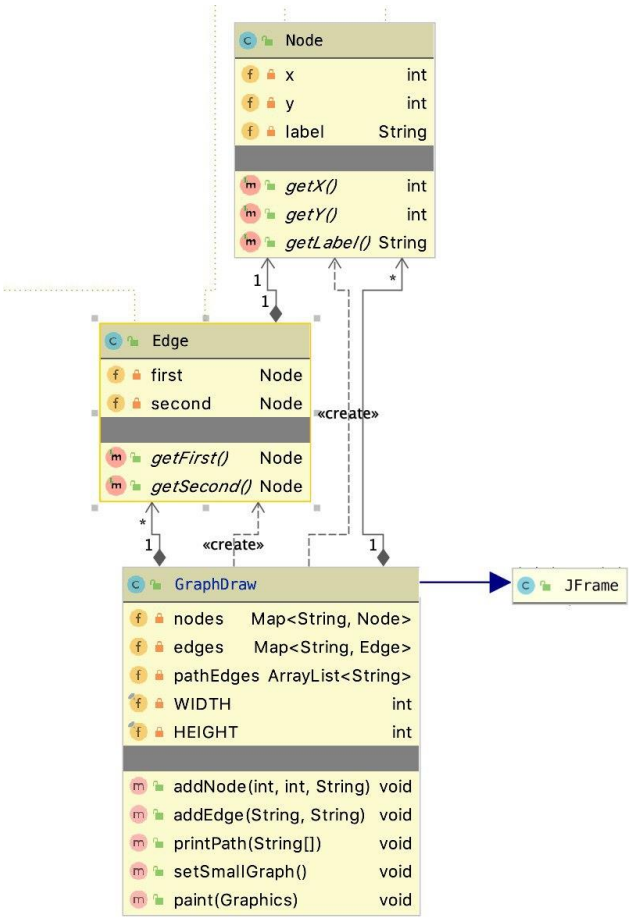


Рис. 3.3. UML діаграма класів графічного відображення графу

3.3 Модель користувацького інтерфейсу

Користувацький інтерфейс складається з двох активних панелей: керуючої та графічної. Головною функцією керуючої панелі є створення можливості для користувача ввести довільний граф відповідно до власних потреб, запуск алгоритму конструювання трафіку та вивід шуканого маршруту. Графічна панель призначена візуалізації заданої системи, а саме виводу графу та подальше виділення оптимальних маршрутів графа.

Для встановлення вершини графа необхідно вказати його положення в декартовій системі координат *(x, y)* та назву вершини. Для цього стеку завдань використовується перший блок команд. На рисунку 3.4 зображено результат задавання трьох вершин графа в межах графічної панелі та дані для встановлення третьої вершини в першому блоці керуючої панелі, що виділено червоною лінією.

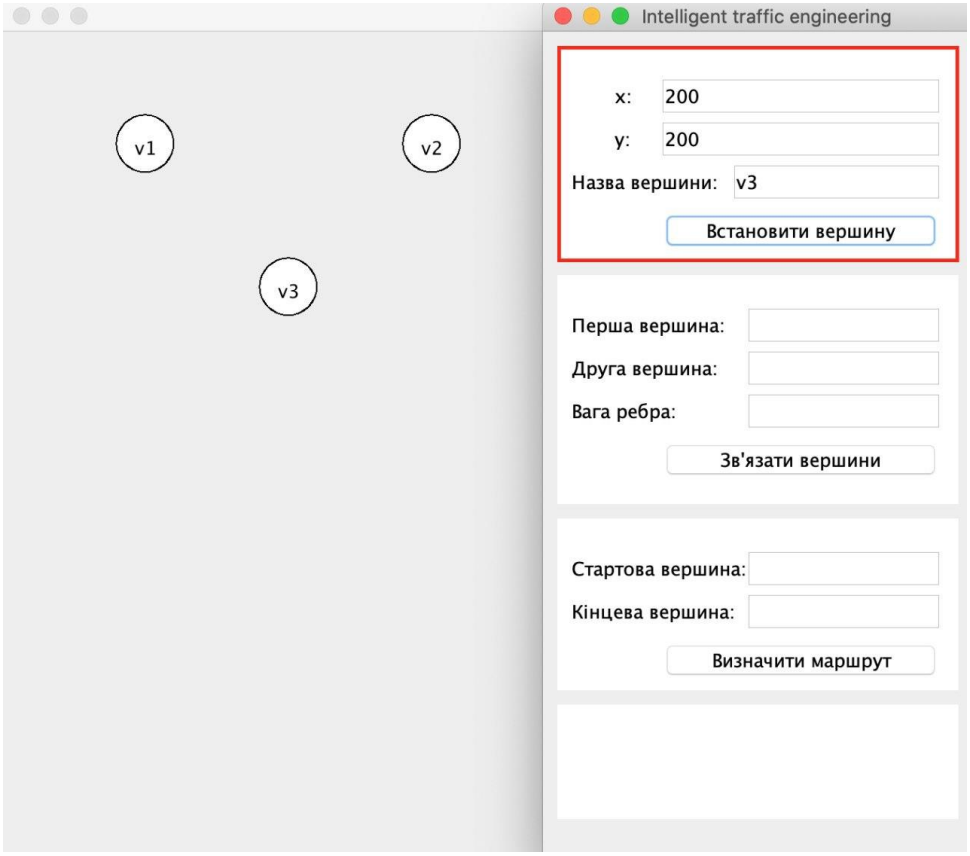


Рис. 3.4. Встановлення вершин графу

Для з’єднання вершин використовується другий блок команд керуючої панелі. Встановлення ребра вимагає визначення вершин, що мають бути зв’язані та вага ребра. На рисунку 3.5 червоним прямокутником виділено обговорюваний блок команд керуючої панелі, та введено дані для встановлення ребра між першою та третьою вершинами.

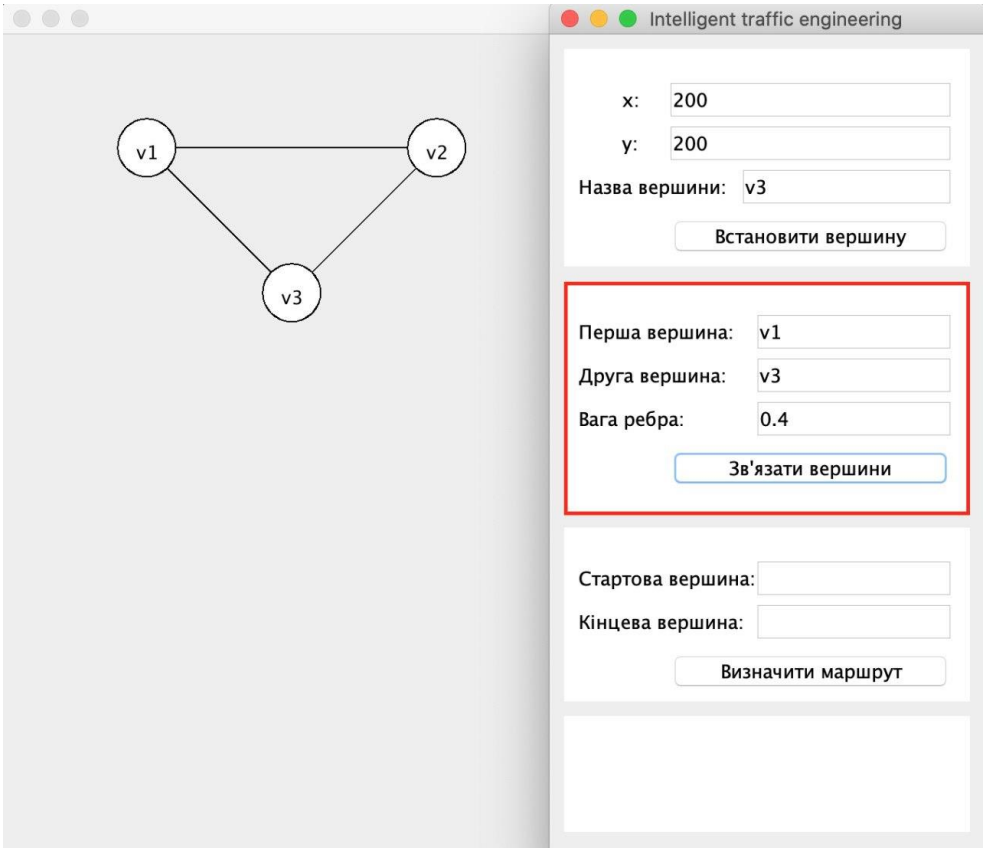


Рис. 3.5. Встановлення ребер графу

Пошук шляху здійснюється по введеним значенням стартової та кінцевої вершини третього блоку команд керуючої моделі. Вивід оптимального шляху, як результат роботи алгоритму, буде здійснено у четвертому блоці даної панелі. Результат також буде виділено червоною лінією на графі системи.

На рисунку 3.6 зображено довільний граф на невелику кількістю вершин та знайдено шлях між ними. Шуканий оптимальний шлях між вершинами підсвічено червоним. Синім блоком виділено панель встановлення

стартової та кінцевої вершини, а червоним блоком виділено вивід шуканого оптимального маршруту між цими вершинами.

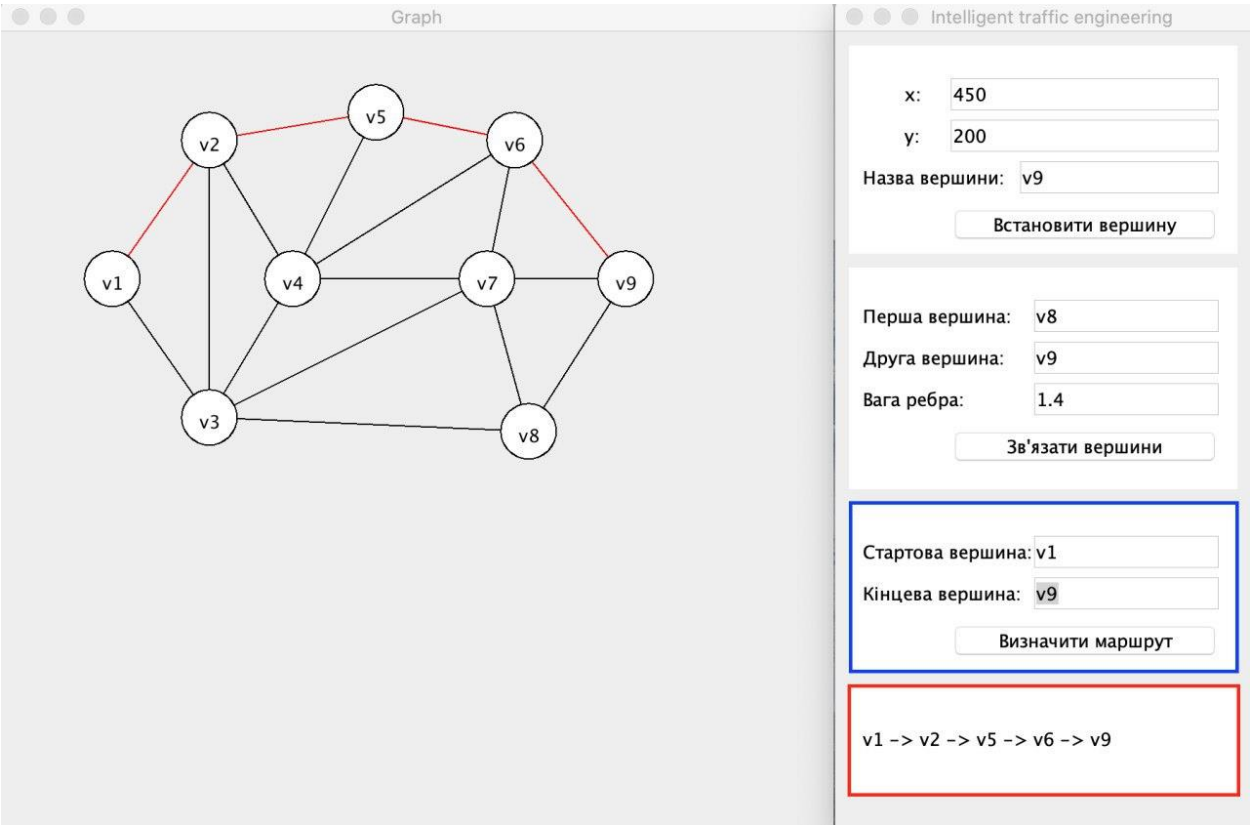


Рис. 3.6. Вивід результатів роботи алгоритму довільного графа

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було наведено детальний опис системи та використаних технологій для її безпосереднього створення. Мовою написання програми було обрано мову об’єктно орієнтовного програмування java; до основних фреймворків відносяться Spring та Hibernate; у якості баз даних було обрано PostgreSQL. Основними критеріями вибору технологій є такі характеристики як: надійність, кросплатформеність, швидкість та простота використання.

Було проведено детальний опис програмного додатку. Головними структурами є граф у якості моделі системи та набір сервісів для симуляції транспортної мережі та її подальшого аналізу для пошуку оптимального маршруту між вершинами.

Графічний інтерфейс програмного додатку складається з керуючої панелі, призначеної для задання вершин графу та зв’язків між ними і графічної панелі, на якому можна побачити стан транспортної мережі та результуючі маршрути між вузлами. Запуск запропонованого алгоритму конструювання трафіку здійснюється в межах керуючої панелі графічного інтерфейсу програми. Також було наведено детальну покрокову інструкцію використання програмного додатку під час задання нового графа та запуску системи.

До виключних ситуацій роботи з інтерфейсом належать лише некоректність введення вхідних даних, але у такому випадку відповідальність лежить саме на користувачеві. Оскільки назва вершини може бути довільною, основною задачею для побудови системи та запуску її аналізу важливо коректно зазначити зв’язки між вузлами. У цьому плані запрограмована транспортна система є універсальною.

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО МОДУЛЮ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Тестування алгоритму конструювання трафіку

Для тестування алгоритму конструювання трафіку буде використовуватись граф, що зображений на рисунку 2.1 попереднього розділу. Для встановлення його вершин та зв'язків між ними було створено окремий метод класу *GraphDraw,* тому ручне введення вершин та ребер (у даному випадку) не є необхідним.

Для пошуку оптимального маршруту буде вибрано вершини v_1 та v_{18}, оскільки вони найвіддаленішими одна від одної та у результаті аналізу трафіку буде зібрані дані з усієї системи. На рисунку 4.1 зображено результуючий граф, з оптимальним маршрутом для заданих вершин, що виділено червоною лінією. Шуканий шлях також можна побачити внизу керуючої панелі. На цьому рисунку немає записів про встановлення вершин та зв'язків між ними, оскільки граф був створений автоматично при старті роботи програми, за допомогою методу *setMainGraph()* класу *GraphDraw.*

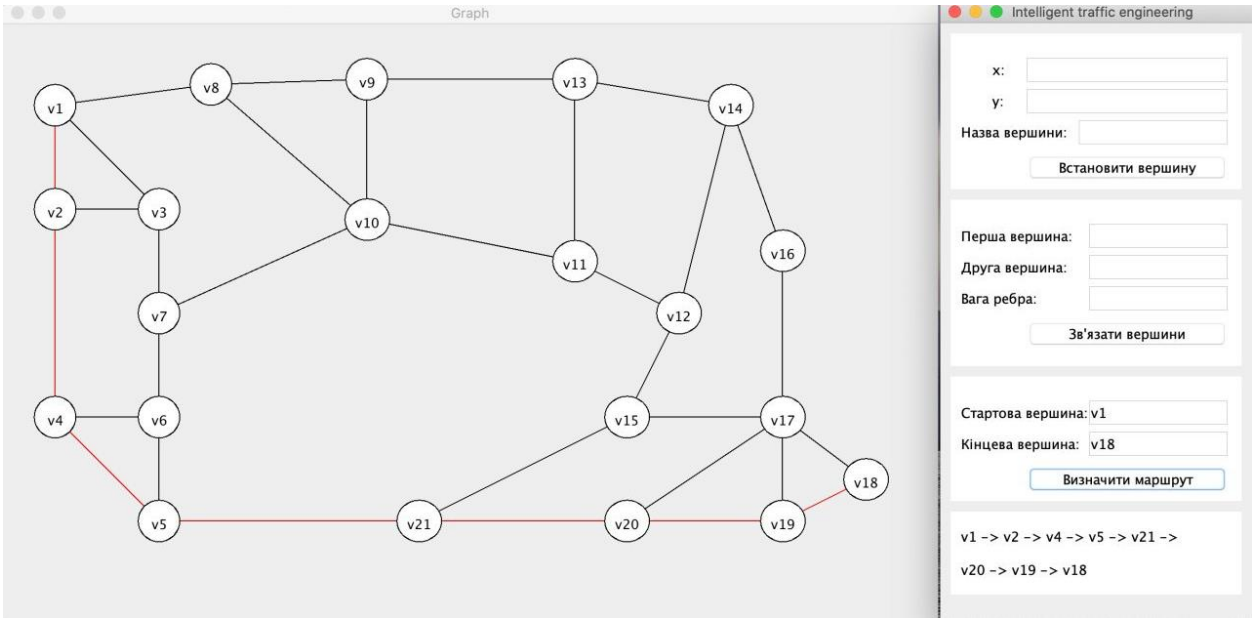


Рис. 4.1. Результат роботи програмного додатку конструювання трафіку інтелектуальної транспортної мережі

Отриманий результат має певні відмінності від теоретичного розрахунку: шлях від 4 до 5 вершини теоретично мав включати шосту вершину, хоча у ході експерименту ці вершини були з'єднані напряму. Така розбіжність є цілком допустимою адже сумарна метрика каналів першого і другого шляху є однаковою, а отже результат тестування є задовільним.

Також у процесі роботи основного алгоритму буде сформовано маршрутну інформацію у таблицях бази даних. У таблиці 4.1 наведено інформацію маршрутизації для першого контролера. Управління наведеною інформацією виконується через спеціальні SQL запити.

Таблиця 4.1. Таблиця маршрутизації контролера S_1

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
1	v18	v8	2.2	105.194973857	v1 → v8 → v9 → v13 → v14 → v16 → v17 → v18
1	v18	v2	1.8	83.9445704207	v1 → v2 → v4 → v5 → v21 → v20 → v19 → v18
1	v18	v3	1.9	78.9055297783	v1 → v3 → v7 → v6 → v5 → v21 → v20 → v19 → v18

З даної таблиці видно, що головним критерієм вибору оптимального маршруту була саме метрика шляху. У межах цієї системи різниця відстані в 0.1 є відносно великим показником, у той час як різниця завантаженості шляху величиною в 3 цілих, є майже невідчутною. Беручи до уваги ці умови, можна зробити висновок, що реалізація алгоритму спрацювала вірно та було отримано правильні, цілком логічні результати.

Наступним кроком експериментального тестування є пошук другого оптимального шляху базуючись на результатах попереднього аналізу системи. Наступним шуканим маршрутом буде шлях від v_7 до v_{14} вершини. Другий маршрут обрано таким чином, щоб відстежити роботу програмного додатку відповідно до другого сценарію. При такому розкладі, для визначення оптимального шляху немає необхідності проведення повторного аналізу системи, адже після пошуку вузли v_7 до v_{14} є проміжними вершинами по відношенню до v_1 до v_{18}. Запуск пошуку маршруту відбувається аналогічно до попереднього. Заповнюються дані про стартову та кінцеві вершини у третьому блоці керуючої панелі, та натискається кнопка “Визначити маршрут”.

На рисунку 4.2 зображено результат роботи додатку у вигляді графу з двома шуканими оптимальними маршрутами, що підсвічені червоним. В нижній частині керуючої панелі виведено результат пошуку другого оптимального маршруту.

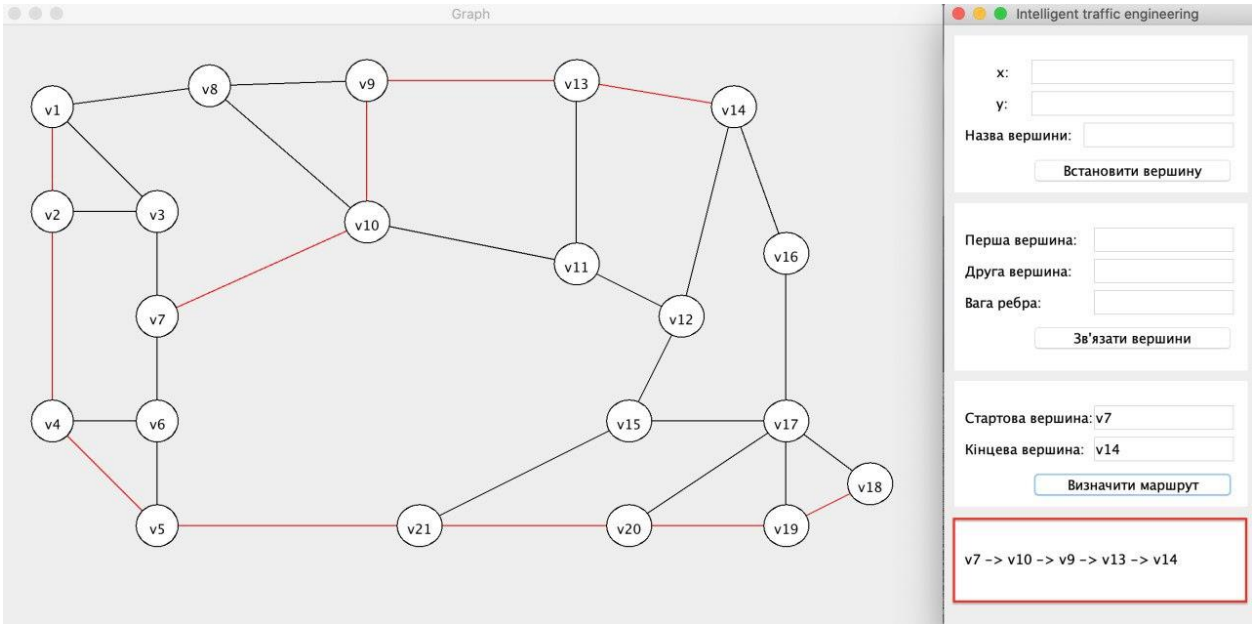


Рис.4.2. Результат пошуку двох оптимальних маршрутів за допомогою програмного додатку конструювання трафіку інтелектуальної транспортної мережі

Для перевірки роботи алгоритму варто знову повернутись до таблиць бази даних системи. Необхідно розглянути саме таблицю маршрутизації сьомого контролера до старту другого пошуку та після його завершення. У таблиці 4.2 відображено дані з бази даних про маршрутну інформацію сьомого контролера до початку пошуку першого оптимального маршруту.

Таблиця 4.2. Таблиця контролера S_7 після пошуку першого шляху

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
7	v18	v10	2.8	86.8061224489	**v7 → v10 → v9 → v13 → v14** → v16 → v17 → v18
7	v18	v6	1.5	80.3279012345	v7 → v6 → v5 → v21 → v20 → v19 → v18

Аналізуючи наведену вище таблицю, водно, що вона зберігає маршрути від сьомої вершини до вісімнадцятої. Кінцевий вузол маршруту був встановлений під час першого пошуку оптимального шляху.

Слід звернути увагу на перший запис таблиці маршрутизації. Жирним шрифтом виділено частину шляху яка вже є результатом другого пошуку. За умови, що система не змінювалась, у процесі пересування транспортних засобів, тобто запити були створені майже одночасно, для вирішення поставленої задачі достатньо просто дістати вже існуючу інформацію з бази даних. Очевидно, що для нового запису також необхідно зробити перерахунок метрики шляху та загруженості каналів. Результатом пошуку буде нова стрічка інформації у таблиці маршрутизації сьомого комутатора. Отриману таблицю з бази даних можна побачити у таблиці 4.3.

Таблиця 4.3. Таблиця маршрутизації контролера S_7

ID контролера	Кінцевий вузол	Суміжна вершина	Метрика шляху	Загрузка шляху	Шлях
7	v18	v10	2.8	86.8061224489	v7 → v10 → v9 → v13 → v14 → v16 → v17 → v18
7	v18	v6	1.5	80.3279012345	v7 → v6 → v5 → v21 → v20 → v19 → v18
7	v10	v7	1.9	30.653739612	v7 → v10 → v9 → v13 → v14

Таким чином у таблиці маршрутизації буде створено запис про новий маршрут, уникнувши витрат часу та ресурсів на повторний аналіз системи. Після створення запису загруженість системи буде обов'язково оновлена. Надійність збереження актуальності даних залишається високою, оскільки система постійно оновлюється.

4.2 Аналіз отриманих результатів

Головною перевагою використання запропонованого методу є отримання найактуальнішої інформації для формування оптимального маршруту між вершинами, демонструючи зниження часу конструювання трафіку та зниження показника часової складності. Як описувалось вище, досягнення таких результатів стає можливим за рахунок збереження маршрутної інформації у базі даних та її постійне оновлення.

Для аналізу результатів роботи алгоритму та оцінки прогресу в плані розробки, було проведено ряд тестів. Основним критерієм порівняння виступають час роботи алгоритму, тобто час формування конструювання

трафіку та величина часової складності алгоритму відповідно до запиту на пошук маршруту.

Під час тестування послідовно проводився пошук оптимального шляху між двома вершинами системи, граф якої зображено на рисунку 2.1. При цьому відстань між вершинами поступово скорочувалась, орієнтуючись не на проміжну кількість вершин, а на кількість рівнів, тобто множин суміжних вершин. Поряд із основним пошуком, проводився пошук маршруту між проміжними вершинами по відношенню до першого шуканого маршруту. Таким чином аналізуючи отримані дані можна побачити дійсно було отримано кращі показники роботи системи завдяки реалізації запрограмованого алгоритму. На рисунку 4.3 зображено графік залежності часу роботи алгоритму від віддаленості вершин одна від одної. Синіми блоками позначено час виконання програми за перший пошук, червоними блоками позначено час виконання програми за другий пошук.

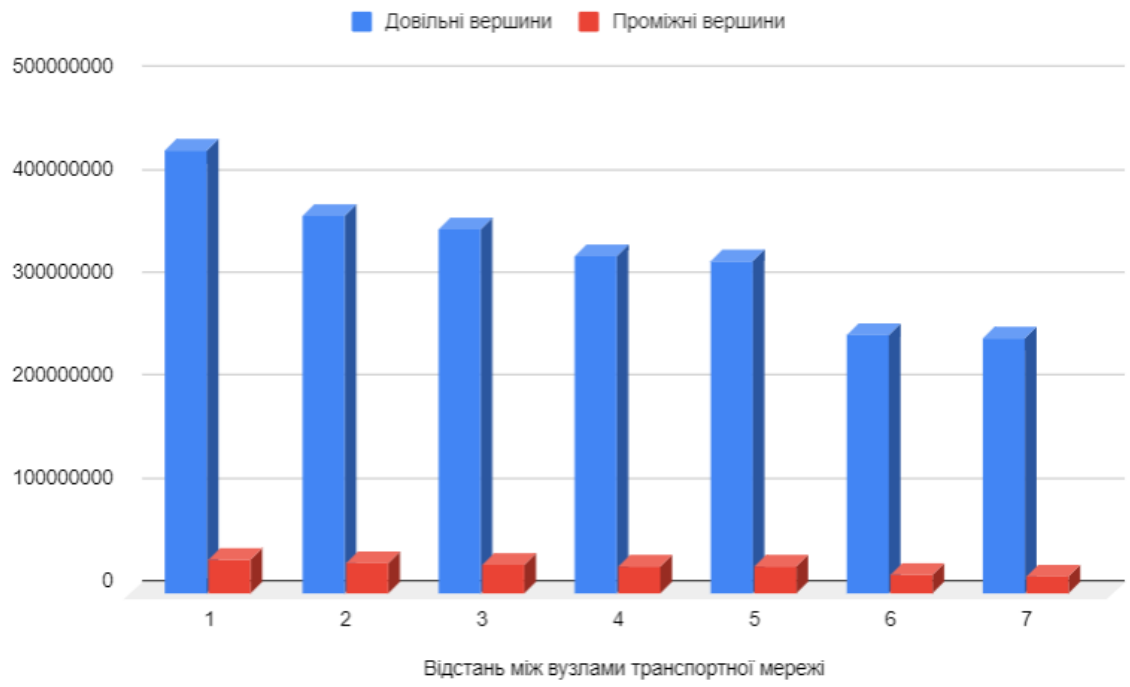


Рис. 4.3. Залежність часу виконання алгоритму конструювання трафіку від відстані між початковою та кінцевою вершинами

По даному графіку видно тенденцію зменшення часу роботи алгоритму з поступовим скороченням відстані між вершинами, не залежно від порядкового номеру проведення пошуку. Таким чином можна зробити твердження, що: час роботи запропонованого алгоритму конструювання трафіку прямопропорційний до відстані між вершинами. Дійсно вражаючою є величина зменшення часу роботи алгоритму при повторному пошуку маршруту для проміжних вершин. Таким чином у результаті вдалось досягти збільшення швидкості виконання програми, що і було основною метою за сталої точності.

Залежність значення часової складності від кількості проведених пошуків маршрути можна побачити на рисунку 4.4.

Рис. 4.4. Графік залежності часової складності алгоритму конструювання трафіку від відстані між початковою та кінцевою вершинами

На даному графіку можна спостерігати схожу тенденцію, що і на попередньому. У результаті аналізу можна зробити два твердження: часова складність запропонованого алгоритму конструювання трафіку

прямопропорційна відстані між заданими вершинами; часова складність роботи алгоритму зменшується за умови проходження попереднього шляху та збереження маршрутної інформації у базі даних.

Розрахунок показників швидкість виконання програми, часова складність роботи алгоритму, точність алгоритму, тощо, я надзвичайно важливим. Адже забезпечуючи надійність роботи системи, точність та швидкість її виконання ми отримуємо меншу кількість матеріальних витрат та меншу кількість аварійних ситуацій, як в обладнанні так і на дорогах, оскільки мова йде саме про транспортну мережу.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі було проведено тестування алгоритмів конструювання трафіку та формування маршрутної інформації системи. Для отримання більш точних та детальних результатів, аналіз мережі проводився за різними, раніше описаними, сценаріями.

Під час моделювання мережі було створено два запити, один за одним, на побудову оптимального маршруту між заданими вузлами. Вважалося, що стан транспортної системи не змінювався, тобто мережа залишалась статичною протягом виконання цих дій. Обидва шляхи розкривають різні аспекти роботи системи, базуючись на різних сценаріях. Аналізуючи отримані результати конструювання трафіку мережі у базі даних та на графічній панелі графічного інтерфейсу, можна зробити висновок, що програма працює вірно. Розраховані дані відповідали теоретичним значенням визначеним у другому розділі. По межами написання документації також було проведено численні тести, створюючи різнопланову нагрузку на систему. У процесі додаткового тестування збоїв роботи програмного додатку не виявлено.

Під час проведення оцінки часу виконання та часової складності запропонованого алгоритму конструювання трафіку було виявлено покращення показників швидкості роботи системи за зниження показників часової складності алгоритму.

Таким чином, використовуючи запропонований у даній роботі алгоритм конструювання трафіку, можна збільшити надійність системи та швидкість надання актуальної інформації, зменшити кількість матеріальних витрат та аварійних ситуацій.

ВИСНОВКИ

Дану роботу присвячено вирішенню проблеми конструювання трафіку в інтелектуальних транспортних мережах. Було наведено детальний опис існуючих методів маршрутизації інтелектуальних мереж з використанням технології SDN. Аналізуючи програмно-визначені транспортні мережі, їх переваги та недоліки, було запропоновано алгоритм конструювання трафіку, що допомагає вирішити наступні проблеми:

1) створити умови автоматичного налагоджування системи для отримання актуальних даних;
2) підвищення швидкості та точності транспортування;
3) зниження кількості аварійних ситуацій для обладнання та безпосередньо на дорогах.

Для вирішення цієї проблеми було запропоновано альтернативний алгоритм конструювання трафіку. В його основі лежить формування маршрутної інформації та вибору оптимального маршруту, базуючись на показниках загруженості мережі да метрики шуканого шляху. Основними відмінностями запропонованого алгоритму від уже існуючих є збереження даних про маршрутизацію у базі даних з подальшим їх використанням для конструювання трафіку та автоматичне оновлення показників, що характеризують трафік.

Інтелектуальна транспортна мережа була представлена у вигляді зваженого графа, вузли якого відповідали окремим транспортним засобам, а ребра — відстані між ними (у якості метрики можна вибрати будь-який показник). Курування та аналіз графу системи відбувається з використанням графічного інтерфейсу, що створює можливість задання нового графу та запуску пошуку оптимального шляху від стартової до кінцевої вершини. У результаті користувач має змогу побачити графічне відображення шуканого маршруту на графічній панелі або отримати результат роботи програми у строковому варіанті в останньому блоці керуючої панелі.

Результат роботи запропонованого алгоритму демонструє виконання головних умов, поставлених на початку розробки програмного додатку. Було проведено детальне тестування окремих модулів та усієї системи в цілому для отримання більш точної оцінки алгоритму. Отримані значення завжди збігались із теоретично розрахованими, не залежно від типу виключної ситуації стану мережі. Різного типу збоїв під час роботи не прослідковувалось.

Для більш детального аналізу алгоритму також було проведено тестування з підрахунком часу виконання запропонованого алгоритму конструювання трафіку та часової складності. У результаті було сформовано основні твердження про роботу алгоритму:

1) точність роботи алгоритму не залежить від віддаленості вершин та від кількості проведених пошуків;
2) час виконання та часова складність запропонованого алгоритму прямопропорційна відстані між заданими вершинами;
3) час виконання та часова складність запропонованого алгоритму зменшується за наявності актуальної маршрутної інформації про проміжні вершини попереднього шляху;
4) час виконання та часова складність запропонованого алгоритму не є пропорційно залежною від кількості пошуків маршруту, за виключенням другого пошуку.

Базуючись на отриманих результатах, можна зробити висновок, що поставлене завдання було виконано у повному обсязі, дотримуючись усіх поставлених умов.

Запрограмована система була створена, як дуже надійна система у якій при транспортуванні даних від одного транспортного засобу не буде виникати виключних ситуацій по перериванню зв’язку. Тому для вдосконалення існуючого рішення необхідно враховувати можливість виходу з ладу транспортних вузлів та забезпечення швидкого надання альтернативного шляху у разі пошкодження цілісності транспортної мережі. Крім того, за

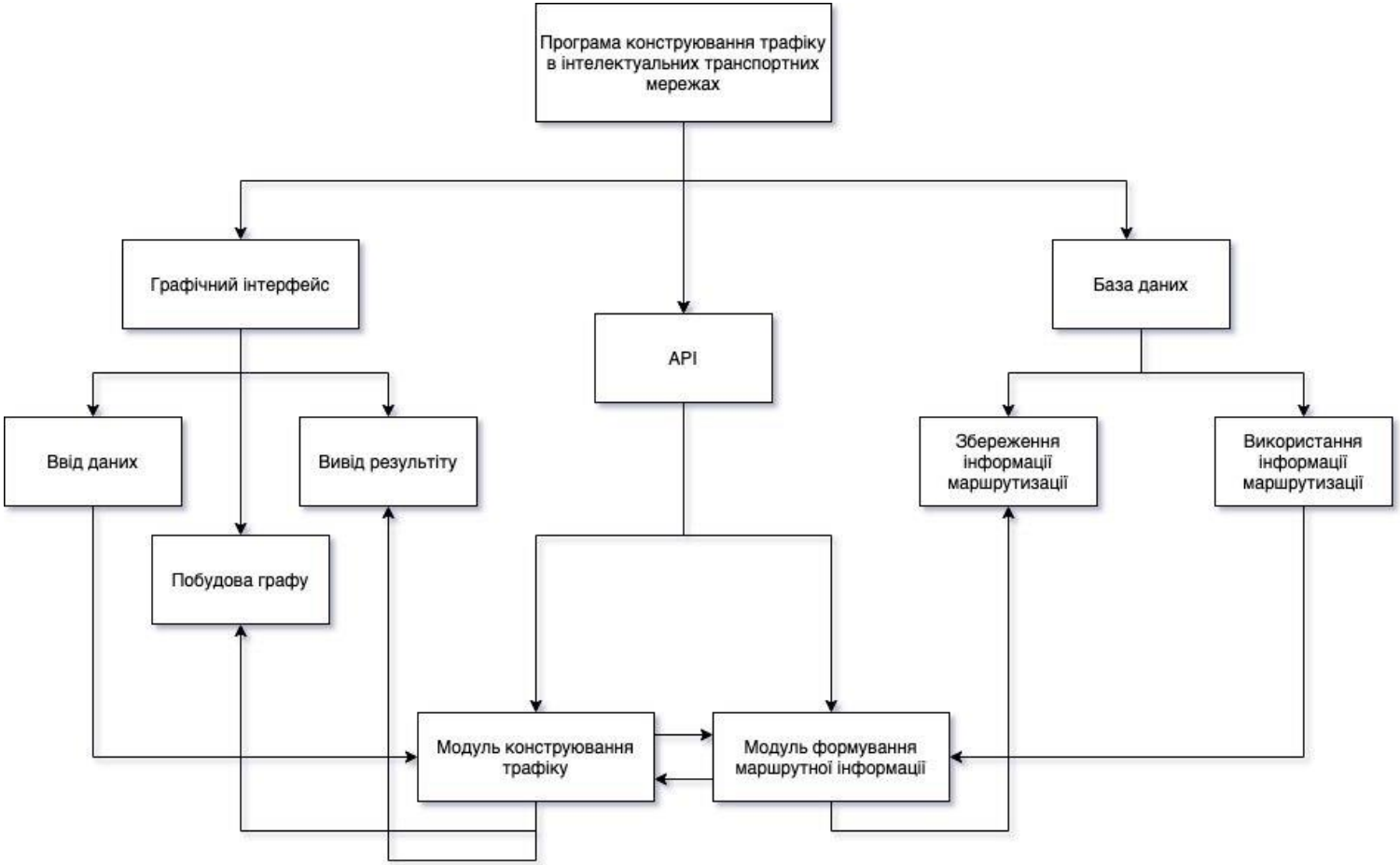
допомогою використання технології GPS стає можливим налаштувати прогнозування виникнення виключних ситуацій та пошкодження трафіку, базуючись на маршруті самого транспортного засобу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Auto Industry Drives Canada’s Industrial Revival– Auto Parts Employment Outpaces Growth In The United States Volume 764 May 2019 Pages 53-62 C. Gomes
2. Intelligent Vehicular Networks and Communications Volume 986 August 2019 Pages 37-44 Anand Paul, Naveen Chilamkurti, Alfred Daniel, Seungmin Rho
3. Definition of Software-Defined Networking (SDN) [Електронний ресурс] // OpenNetworkingFoundation – Режим доступу до ресурсу: https://www.opennetworking.org/sdn-definition/
4. Kumar, Arun & Shwe, Hnin & Wong, Kai & Chong, P.H.J.. (2017). Location-Based Routing Protocols for Wireless Sensor Networks: A Survey. Wireless Sensor Network. 9. 25-72. 10.4236/wsn.2017.91003.
5. A wave propagation-based adaptive probabilistic broadcast containment strategy for reactive MANET routing protocols Pervasive and Mobile Computing Volume 2017 Pages 628-638
6. ODCR: Energy Efficient and Reliable Density Clustered-based routing protocol for emergency sensor applications Applied Computing and InformaticsIn press, corrected proof Available online 25 March 2020 Mohammed S. Al-kahtani, Lutful Karim, Nargis Khan
7. Prominent unicast routing protocols for Mobile Ad hoc Networks: Criterion, classification, and key attributes Ad Hoc Networks Volume 891 June 2019 Pages 58-77 Trilok Kumar Saini, Subhash C. Sharma
8. Organized topology based routing protocol in incompletely predictable ad-hoc networks Computer Communications Volume 991 February 2017 Pages 107-118 Jian Shen, Chen Wang, Anxi Wang, Xingming Sun, Patrick C. K. Hung
9. Performance Analysis of AODV and DSDV Routing Protocol in MANET and Modifications in AODV against Black Hole Attack Procedia Computer Science Volume 79 2016 Pages 835-844 A. A. Chavan, D. S. Kurule, P. U. Dere

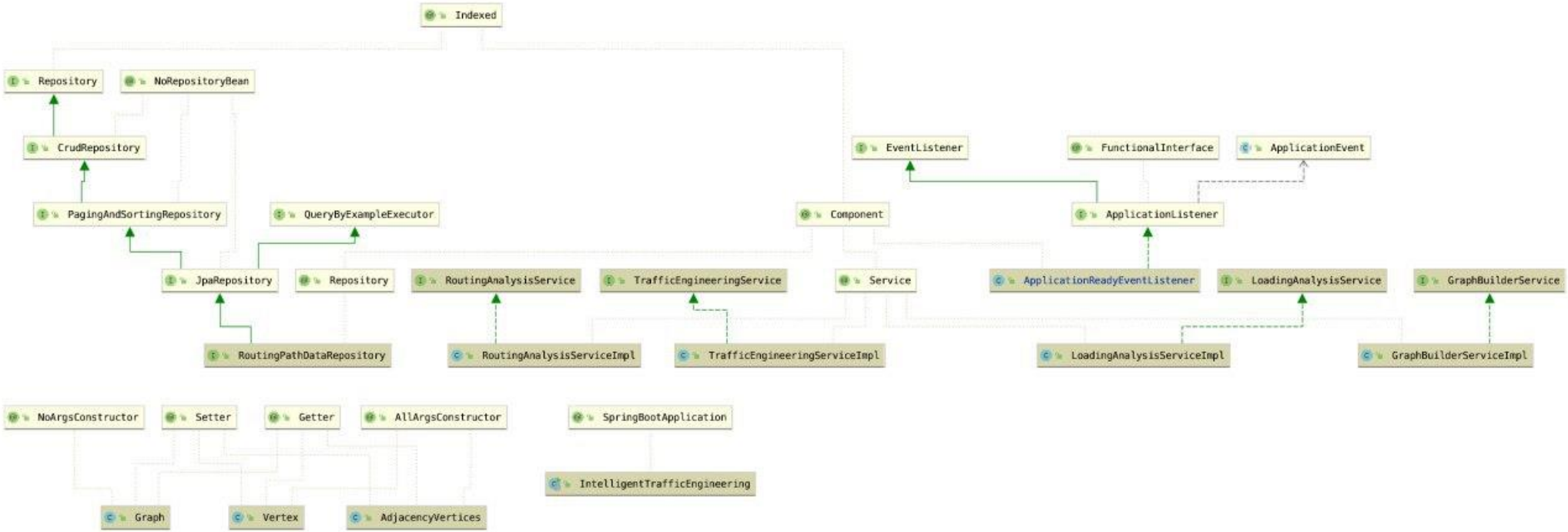
10. Effects of Velocity on Performance of DYMO, AODV and DSR Routing Protocols in Mobile Ad-hoc Networks Procedia Technology Volume 42012 Pages 727-731 Lakhan Dev Sharma, Nirmal Roberts
11. A Hybrid Routing Protocol for VANET Using Ontology Procedia Computer Science Volume 73 2015 Pages 94-101 Hamza Toulni, Benayad Nsiri
12. SSDN Architecture Issue 1.0 2 ONF Document Type: TR (Technical Reference), non-normative, type 2 ONF Document Name: SDN ARCH 1.0 06062014
13. Energy-efficient multicast routing protocol based on SDN and fog computing for vehicular networks Ad Hoc Networks Volume 841 March 2019 Pages 68-81 Ahmed Jawad Kadhim, Seyed Amin Hosseini Seno
14. Controller placement optimization in hierarchical distributed software defined vehicular networks Computer Networks Volume 13522 April 2018 Pages 226-239 Kushan Sudheera Kalupahana Liyanage, Maode Ma, Peter Han Joo Chong
15. Efficient data dissemination in cooperative multi-RSU Vehicular Ad Hoc Networks (VANETs) Journal of Systems and Software Volume 117 July 2016 Pages 508-527 G. G. Md. Nawaz Ali, Peter Han Joo Chong, Syeda Khairunnesa Samantha, Edward Chan
16. Clustered robust routing for traffic engineering in software-defined networks Computer Communications Volume 14415 August 2019 Pages 175-187 Davide Sanvito, Ilario Filippini, Antonio Capone, Stefano Paris, Jérémie Leguay\
17. Vehicular software-defined networking and fog computing: Integration and design principles Ad Hoc NetworksVolume 82January 2019 Pages 172-181 Jéferson Campos Nobre, Allan M. de Souza, Denis Rosário, Cristiano Both, Mario Gerla

ДОДАТОК 1. Структурна схема пристрою



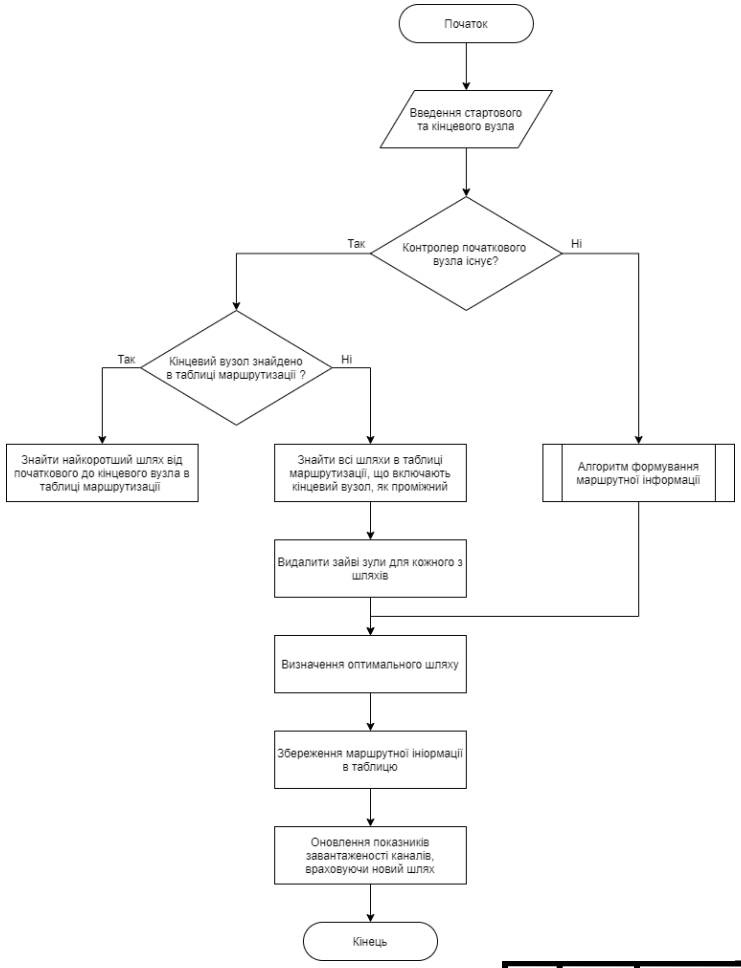
					ІАЛЦ.466454.004.Д1		
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 1. Структурна схема пристрою	Літера	Маса / Масштаб
Розробив		Осієвська В. С.					
Перевірив		Калюжний О. О.				Лист 1	Листів 1
Т. контр							
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-61	
Затв.							

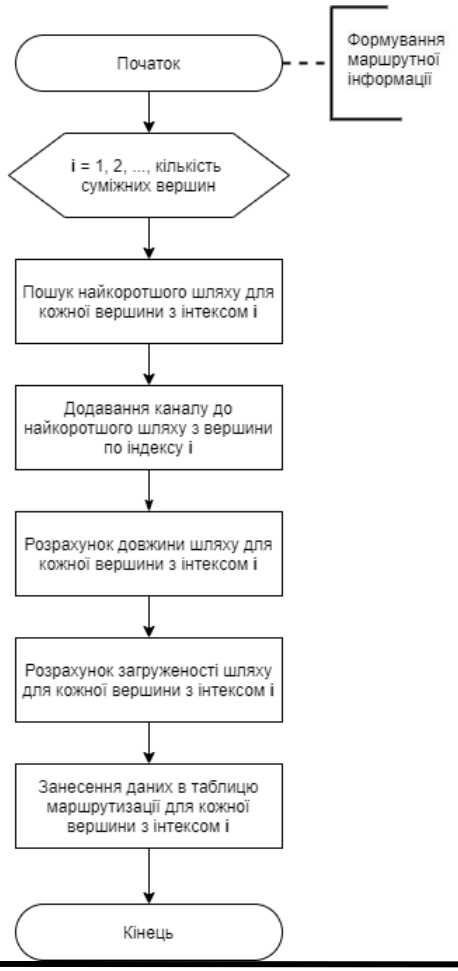
ДОДАТОК 2. Функціональна схема



					ІАЛЦ.466454.005.Д2	
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 2. Функціональна Схема	Літера / Маса / Масштаб
Розробив		Осієвська В. С.				
Перевірив		Калюжний О. О.				Лист 1 / Листів 1
Т. контр						
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-61
Затв.						

ДОДАТОК 3. Принципова схема апаратного забезпечення





					ІАЛЦ.466454.006.Д3	
Зм.	Арк.	№ документа	Підпис	Дата	ДОДАТОК 3. Принципова схема апаратного забезпечення	Літера / Маса / Масштаб
Розробив		Осієвська В. С.				
Перевірив		Калюжний О. О.				Лист 1 / Листів 1
Т. контр						
Н.контр.		Сімоненко В. П.				ФІОТ Гр. ІО-61
Затв.						

ДОДАТОК 4. Програмний код додатку

```
package com.valentyna.intelligent.traffic.engineering.service;

import com.valentyna.intelligent.traffic.engineering.graph.Vertex;

public interface RoutingAnalysisService {

   void performRoutingDataAnalysis(Vertex source, Vertex destination);

   int getTotalOperationAmount();
}
package com.valentyna.intelligent.traffic.engineering.service.impl;

import com.valentyna.intelligent.traffic.engineering.domen.RoutingPathData;
import com.valentyna.intelligent.traffic.engineering.graph.AdjacencyVertices;
import com.valentyna.intelligent.traffic.engineering.graph.Graph;
import com.valentyna.intelligent.traffic.engineering.graph.Vertex;
import com.valentyna.intelligent.traffic.engineering.repository.RoutingPathDataRepository;
import com.valentyna.intelligent.traffic.engineering.service.GraphBuilderService;
import com.valentyna.intelligent.traffic.engineering.service.RoutingAnalysisService;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

@Service
public class RoutingAnalysisServiceImpl implements RoutingAnalysisService {

   private Graph graph;
   private Vertex source;
   private Vertex destination;
   private List<Vertex> queue = new ArrayList<>();
   private int totalOperationsAmount;

   private final RoutingPathDataRepository routingDataRepository;

   public RoutingAnalysisServiceImpl(RoutingPathDataRepository routingDataRepository,
                                     GraphBuilderService graphBuilderService) {
```

```
        this.routingDataRepository = routingDataRepository;
        this.graph = graphBuilderService.buildGraph();
        this.totalOperationsAmount = 0;
    }

    public void init(Vertex source, Vertex destination) {
        this.source = source;
        this.destination = destination;
    }

    @Override
    public void performRoutingDataAnalysis(Vertex source, Vertex destination) {
        init(source, destination);

        AdjacencyVertices firstVertices = getFirstAdjacencyVertices(); // step 2 [create first
vertices layer]
        AdjacencyVertices nextVertices = getNextAdjacencyVertices(firstVertices);

        nextVertices.getCurrentVertices().forEach(v ->
routingDataRepository.save(createRoutingPathData(v, this.destination))); // step 3 [fill first
tables with data]

        for (Vertex currentVertex : nextVertices.getCurrentVertices()) { // step 4 [exchange of
routing information between switches of the same level]
            routingDataRepository.saveAll(gatherRoutingPathInformation(nextVertices,
currentVertex));
        }

        while (sourceNotReached(nextVertices)) {
            nextVertices = getNextAdjacencyVertices(nextVertices);

            for (Vertex currentVertex : nextVertices.getCurrentVertices()) {
                gatherRoutingPathInformation(nextVertices,
currentVertex).forEach(routingDataRepository::save);

            }

            while (!queue.isEmpty()) {
                Vertex currentVertex = queue.get(0);
                gatherRoutingPathInformation(nextVertices,
currentVertex).forEach(routingDataRepository::save);
                queue.remove(currentVertex);
            }
        }
    }

    @Override
```

```
    public int getTotalOperationAmount() {
        int totalOperationsAmount = this.totalOperationsAmount;
        this.totalOperationsAmount = 0;
        return totalOperationsAmount;
    }


    private List<RoutingPathData> gatherRoutingPathInformation(AdjacencyVertices
adjacencyVertices, Vertex currentVertex) {
        increase();
        List<RoutingPathData> pathInfoList = new ArrayList<>();

        Set<Vertex> possibleAdjVertex = Stream.of(adjacencyVertices.getCurrentVertices(),
adjacencyVertices.getPreviousVertices())
                .flatMap(Collection::stream)
                .filter(v -> !isCurrentVertex(currentVertex, v))
                .filter(v -> !isDestinationVertex(v))
                .filter(v -> isAdjacencyVertex(currentVertex, v))
                .filter(v -> !isVisitedAdjacencyVertex(currentVertex, v))
                .collect(Collectors.toSet());

        increase();
        possibleAdjVertex.stream()
                .filter(v -> tableIsPresent(getControllerId(v)))
                .forEach(v -> pathInfoList.add(createRoutingPathData(currentVertex, v)));

        possibleAdjVertex.stream()
                .filter(v -> !tableIsPresent(getControllerId(v)))
                .forEach(v -> this.queue.add(currentVertex));

        return pathInfoList;
    }

    private RoutingPathData createRoutingPathData(Vertex currentVertex, Vertex adjVertex) {
        increase();
        String path = getShortestPathFromVertexToDestination(currentVertex, adjVertex);
        return RoutingPathData.builder()
                .controllerId(getControllerId(currentVertex))
                .adjacentTop(adjVertex.getLabel())
                .weight(getPathWeight(path))
                .path(path)
                .source(currentVertex.getLabel())
                .destination(destination.getLabel())
                .build();
    }

    private double getPathWeight(String path) {
```

```
        increase();
        String[] vertices = path.split(" -> ");
        return IntStream.range(0, vertices.length - 1)
                .mapToDouble(i -> this.graph.getEdgesWeights().get(vertices[i] + "-" + vertices[i +
1]))
                .sum();
    }

    private String getShortestPathFromVertexToDestination(Vertex currentVertex, Vertex
adjVertex) {
        increase();
        if (isDestinationVertex(adjVertex)) {
            return currentVertex.getLabel() + " -> " + adjVertex.getLabel();
        }

        List<String> result = new ArrayList<>();
        result.add(currentVertex.getLabel());

        Set<RoutingPathData> routingPathDataList =
routingDataRepository.findAllByControllerId(getControllerId(adjVertex));
        return currentVertex.getLabel() + " -> " +
getShortestPathInRoutingTable(routingPathDataList);
    }

    private String getShortestPathInRoutingTable(Set<RoutingPathData> routingPathDataList) {
        increase();
        return routingPathDataList.stream()
                .min(Comparator.comparing(RoutingPathData::getWeight))
                .get()
                .getPath();
    }

    private boolean tableIsPresent(int adjControllerId) {
        return !routingDataRepository.findAllByControllerId(adjControllerId).isEmpty();
    }

    private AdjacencyVertices getFirstAdjacencyVertices() {
        increase();
        return new AdjacencyVertices(1, Set.of(this.destination), new HashSet<>());
    }

    private AdjacencyVertices getNextAdjacencyVertices(AdjacencyVertices vertices) {
        increase();
        Set<Vertex> nextVertices = vertices.getCurrentVertices().stream()
                .map(v -> this.graph.getAdjacencyVertices().get(v))
                .flatMap(Collection::stream)
                .filter(v -> !vertices.getCurrentVertices().contains(v))
```

```
            .filter(v -> !vertices.getPreviousVertices().contains(v))
            .collect(Collectors.toSet());

        return new AdjacencyVertices(vertices.getId() + 1, nextVertices,
vertices.getCurrentVertices());
    }

    private boolean isVisitedAdjacencyVertex(Vertex currentVertex, Vertex adjVertex) {
        Set<RoutingPathData> routingPathDataList =
routingDataRepository.findAllByControllerId(getControllerId(currentVertex));

        return routingPathDataList.stream().anyMatch(i ->
i.getAdjacentTop().equals(adjVertex.getLabel()));
    }

    private boolean isAdjacencyVertex(Vertex currentVertex, Vertex vertex) {
        return this.graph.getAdjacencyVertices().get(currentVertex).contains(vertex);
    }

    private boolean isDestinationVertex(Vertex adjVertex) {
        return adjVertex.equals(this.destination);
    }

    private boolean isCurrentVertex(Vertex currentVertex, Vertex adjVertex) {
        return adjVertex.equals(currentVertex);
    }

    private boolean sourceNotReached(AdjacencyVertices nextVertices) {
        increase();
        return !nextVertices.getPreviousVertices().contains(source);
    }

    private int getControllerId(Vertex vertex) {
        return Integer.parseInt(vertex.getLabel().substring(1));
    }

    private void increase() {
        this.totalOperationsAmount++;
    }
}
package com.valentyna.intelligent.traffic.engineering.service;

public interface LoadingAnalysisService {

    void performLoadingDataAnalysis();

    double calculateLoadCriterion(String path);
```

```
    int getTotalOperationAmount();
}
package com.valentyna.intelligent.traffic.engineering.service.impl;

import com.valentyna.intelligent.traffic.engineering.domen.RoutingPathData;
import com.valentyna.intelligent.traffic.engineering.graph.Graph;
import com.valentyna.intelligent.traffic.engineering.panel.GraphData;
import com.valentyna.intelligent.traffic.engineering.repository.RoutingPathDataRepository;
import com.valentyna.intelligent.traffic.engineering.service.GraphBuilderService;
import com.valentyna.intelligent.traffic.engineering.service.LoadingAnalysisService;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

@Service
public class LoadingAnalysisServiceImpl implements LoadingAnalysisService {

    private final RoutingPathDataRepository routingPathDataRepository;

    private Map<String, Double> chanelLoadings = new HashMap<>();

    private Graph graph;

    private int totalOperationsAmount;

    public LoadingAnalysisServiceImpl(RoutingPathDataRepository routingPathDataRepository,
GraphBuilderService graphBuilderService) {
        this.routingPathDataRepository = routingPathDataRepository;
        graph = graphBuilderService.buildGraph();
    }

    @Override
    public void performLoadingDataAnalysis() {
        chanelLoadings = new HashMap<>();
        calculateChanelLoading();
        List<RoutingPathData> routingPathData = routingPathDataRepository.findAll();
        for (RoutingPathData data : routingPathData) {
            double pathLoading = calculateLoadCriterion(data.getPath());
            data.setLoading(pathLoading);
            routingPathDataRepository.save(data);
        }
```

```
    }

    @Override
    public double calculateLoadCriterion(String path) {
        List<String> channels = getChannels(path);

        List<Double> channelLoading =
channels.stream().map(this::getChannelLoading).collect(Collectors.toList());
        List<Double> channelWeight = channels.stream().map(c ->
graph.getEdgesWeights().get(c)).collect(Collectors.toList());
        double pathWeight = channelWeight.stream().mapToDouble(Double::doubleValue).sum();

        double averageChanelLoading = IntStream.range(0, channels.size())
                .mapToDouble(i -> (channelLoading.get(i) * channelWeight.get(i)) / pathWeight)
                .sum() / channels.size();

        double criteria = IntStream.range(0, channels.size())
                .mapToDouble(i -> (channelWeight.get(i) * Math.pow(channelLoading.get(i) -
averageChanelLoading, 2)) / pathWeight)
                .sum() + averageChanelLoading;

        totalOperationsAmount += channelLoading.size() * 16 * 16 * 32 * 16 * 16 * 64 * 16 * 32
* 16;
        return criteria;
    }

    @Override
    public int getTotalOperationAmount() {
        int totalOperationsAmount = this.totalOperationsAmount;
        this.totalOperationsAmount = 0;
        return totalOperationsAmount;
    }

    private void calculateChanelLoading() {
        routingPathDataRepository.findAll().stream()
                .map(data -> data.getPath().split(" -> "))
                .forEach(path -> IntStream.range(0, path.length - 1)
                        .forEach(i -> updateChanelLoadingData(path[i], path[i + 1])));
    }

    private Double getChannelLoading(String channel) {
        increase();
        return chanelLoadings.get(channel) != null ? chanelLoadings.get(channel)
                : chanelLoadings.get(channel.split("-")[1] + "-" + channel.split("-")[0]);
    }

    private List<String> getChannels(String path) {
```

```
        increase();
        String[] vertices = path.split(" -> ");
        List<String> channels = new ArrayList<>();
        for (int i = 0; i < vertices.length - 1; i++) {
            channels.add(vertices[i] + "-" + vertices[i + 1]);
        }
        return channels;
    }

    private void updateChanelLoadingData(String v1, String v2) {
        increase();
        String key = v2 + "-" + v1;

        if (chanelLoadings.containsKey(key)) {
            chanelLoadings.put(key, chanelLoadings.get(key) + 1.0);
        } else {
            key = v1 + "-" + v2;
            if (chanelLoadings.containsKey(key)) {
                chanelLoadings.put(key, chanelLoadings.get(key) + 1.0);
            } else {
                chanelLoadings.put(key, 1.0);
            }
        }
    }

    void increase() {
        totalOperationsAmount++;
    }
}
package com.valentyna.intelligent.traffic.engineering.service;

import com.valentyna.intelligent.traffic.engineering.graph.Vertex;

public interface TrafficEngineeringService {

    String startTrafficEngineering(Vertex source, Vertex destination);
}
package com.valentyna.intelligent.traffic.engineering.service.impl;

import com.valentyna.intelligent.traffic.engineering.domen.RoutingPathData;
import com.valentyna.intelligent.traffic.engineering.graph.Graph;
import com.valentyna.intelligent.traffic.engineering.graph.Vertex;
import com.valentyna.intelligent.traffic.engineering.repository.RoutingPathDataRepository;
import com.valentyna.intelligent.traffic.engineering.service.GraphBuilderService;
import com.valentyna.intelligent.traffic.engineering.service.LoadingAnalysisService;
import com.valentyna.intelligent.traffic.engineering.service.RoutingAnalysisService;
import com.valentyna.intelligent.traffic.engineering.service.TrafficEngineeringService;
```

```
import org.springframework.stereotype.Service;

import java.util.Comparator;
import java.util.Set;
import java.util.stream.IntStream;

@Service
public class TrafficEngineeringServiceImpl implements TrafficEngineeringService {

  private final LoadingAnalysisService loadingAnalysisService;

  private final RoutingAnalysisService routingAnalysisService;

  private final RoutingPathDataRepository routingPathDataRepository;

  private final Graph graph;

  private int totalOperationsAmount = 0;

  public TrafficEngineeringServiceImpl(LoadingAnalysisService loadingAnalysisService,
                                       RoutingAnalysisService routingAnalysisService,
                                       RoutingPathDataRepository routingPathDataRepository,
                                       GraphBuilderService graphBuilderService) {
    this.loadingAnalysisService = loadingAnalysisService;
    this.routingAnalysisService = routingAnalysisService;
    this.routingPathDataRepository = routingPathDataRepository;
    this.graph = graphBuilderService.buildGraph();
  }

  @Override
  public String startTrafficEngineering (Vertex source, Vertex destination) {
    long startTime = System.nanoTime();
    String path = findPathInController(source, destination);

    RoutingPathData routingPathData =
routingPathDataRepository.findByPath(path).orElse(null);

    if (routingPathData == null) {
      totalOperationsAmount++;
      routingPathData = createNewRoutingData(path, source, destination);
      routingPathData = routingPathDataRepository.save(routingPathData);
    }

    loadingAnalysisService.performLoadingDataAnalysis();
    routingPathData.setLoading(loadingAnalysisService.calculateLoadCriterion(path));
    routingPathDataRepository.save(routingPathData);
    totalOperationsAmount += loadingAnalysisService.getTotalOperationAmount();
```

```
//      String result = "[" + path + "]" + " \n1) Time complexity = " + totalOperationsAmount +
Math.log(totalOperationsAmount)
//              + "\n2) Performance time: " + (System.nanoTime() - startTime);
        totalOperationsAmount = 0;
        return path;
    }

    public String findPathInController(Vertex source, Vertex destination) {
        Set<RoutingPathData> routingControllerData =
routingPathDataRepository.findAllByControllerId(getControllerId(source));

        // source controller to destination exist, return shortest path
        RoutingPathData routingPathData = routingControllerData.stream()
                .filter(d -> d.getDestination().equals(destination.getLabel()))
                .min(Comparator.comparing(RoutingPathData::getWeight))
                .orElse(null);

        if (routingPathData != null) {
            totalOperationsAmount++;
            return routingPathData.getPath();
        }

        // if controller exist find destination in controller paths
        String path = routingControllerData.stream()
                .map(RoutingPathData::getPath)
                .filter(p -> p.contains(destination.getLabel()))
                .map(p -> p.substring(0, p.indexOf(destination.getLabel())) + destination.getLabel())
                .min(Comparator.comparing(this::getPathWeight))
                .orElse(null);

        if (path != null) {
            totalOperationsAmount++;
            return path;
        }

        // if source does not exist, perform new analysis and try again
        routingAnalysisService.performRoutingDataAnalysis(source, destination);
        totalOperationsAmount += routingAnalysisService.getTotalOperationAmount();
        return findPathInController(source, destination);
    }

    private RoutingPathData createNewRoutingData(String path, Vertex source, Vertex
destination) {
        String[] vertices = path.split(" -> ");
        return RoutingPathData.builder()
                .controllerId(Integer.parseInt(vertices[0].substring(1)))
```

```
                .adjacentTop(vertices[1])
                .source(source.getLabel())
                .destination(destination.getLabel())
                .path(path)
                .weight(getPathWeight(path))
                .build();
    }

    private int getControllerId(Vertex vertex) {
        return Integer.parseInt(vertex.getLabel().substring(1));
    }

    private double getPathWeight(String path) {
        String[] vertices = path.split(" -> ");
        return IntStream.range(0, vertices.length - 1)
                .mapToDouble(i -> graph.getEdgesWeights().get(vertices[i] + "-" + vertices[i + 1]))
                .sum();
    }
}
```